

DTIC FILE COPY

WRDC-TR-89-204:

AD-A215 134

TURBINE BLADE
DATA ACQUISITION SYSTEM
SOFTWARE REFERENCE

M. J. Gutman

University of Dayton Research Institute
Electronic and Computer Development Laboratory
300 College Park Avenue
Dayton, OH 45469-0001

May 1989

Final Report for Period September 1985 to August 1987

Approved for public release; distribution unlimited.

AERO PROPULSION AND POWER LABORATORY
WRIGHT RESEARCH AND DEVELOPMENT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6563



DTIC
ELECTE
DEC 07 1989
S E D

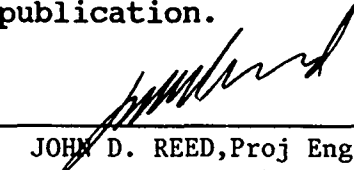
00 12 00 021

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing to the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use or sell any patented invention that may in any way be related thereto.

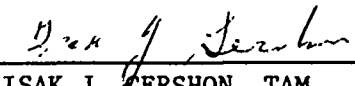
This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.




JOHN D. REED, Proj Engr
Assesment Branch
Turbine Engine Division
Aero Propulsion & Power
Laboratory

FOR THE COMMANDER



ISAK J. GERSHON, TAM
Components Branch
Turbine Engine Division
Aero Propulsion & Power
Laboratory



GEOFFREY W. JUMPER, Maj, USAF
Chief, WRDC/POTC
Turbine Engine Division
Aero Propulsion & Power
Laboratory

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/POTC, WPAFB, OH 45433-6563 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UDR-TR-87-134			5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-89-2042		
6a. NAME OF PERFORMING ORGANIZATION University of Dayton Research Institute		6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Wright Research & Development Center Aero Propulsion & Power Lab, AISC		
6c. ADDRESS (City, State, and ZIP Code) 300 College Park Avenue Dayton, OH 45469		7b. ADDRESS (City, State, and ZIP Code) WRDC/POTC WPAFB, OH 45433-6563			
8a. NAME OF FUNDING / SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Contract F33615-85-C-2585		
8c. ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS			
		PROGRAM ELEMENT NO. 62203F	PROJECT NO. 3066	TASK NO. 12	WORK UNIT ACCESSION NO. 21
11. TITLE (Include Security Classification) Turbine Blade Data Acquisition System (Software Reference)					
12. PERSONAL AUTHOR(S) Gutman, Michael Joseph; Blanchard, Robert Edson; Aulds, James Michael and Shopu, Anthony Lee					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM 9/85 TO 8/87		14. DATE OF REPORT (Year, Month, Day) May 1989	
15. PAGE COUNT 86					
16. SUPPLEMENTARY NOTATION The computer software contained herin are theoretical and/or references that in no way reflect Air Force-owned or-developed computer software. <i>Turbine blades</i>					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Forced Response, Bladed Disk, Dynamic Structural Analysis, Noncontacting Stress Measurement System,		
21	05				
14	02				
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This report contains design and technical documentation on magnetic tape format, master and slave program software, and the simulator EPROM listings for the noncontacting stress measurement system. This system is used as an analytical tool for structural testing and research on bladed disk components. The data acquisition software is used to control the operation of the signal storage and processing electronics. <i>Keywords:</i>					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL John D. Reed, Project Engineer, POTC			22b. TELEPHONE (Include Area Code) (513) 255-2367		22c. OFFICE SYMBOL W C/POTC

This report provides design and technical documentation on the Turbine Blade Data Acquisition System developed by the Electronic and Computer Development Laboratory, within the University of Dayton Research Institute. It was designed and fabricated as an analytical tool for structural testing and research on turbine components as part of the Noncontacting Stress Measurement System. This research effort was performed for the Aerospace Mechanics Group of the Research Institute. Michael Drake was the Principal Investigator and Robert Dominic was the Research Engineer in charge of daily activities for this project.

E&CD Lab personnel who contributed to this research and development effort were:

Lab Supervisor	G. Thomas Collins
Electronic Design	J. Michael Aulds Robert Blanchard
Fabrication and Checkout	Ben Connally Fred L. Davis Steve Fuchs
Documentation	Michael Gutman Steve Fuchs
Report Assembly	Sam Pietrantonio

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input checked="" type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



CONTENTS

1.0	MAGNETIC TAPE FORMAT	1
2.0	COMPILATION CONTROL FILES	3
2.1	rom.com	3
2.2	call.com	3
2.3	aall.com	3
2.4	turbin.l	3
2.5	rom.com	4
2.6	aall.com	4
2.7	slave.l	4
3.0	MASTER PROGRAM SOFTWARE LISTINGS	7
3.1	clockr.s	7
3.2	clockw.s	7
3.3	cpm.s	9
3.4	revers.s	11
3.5	risr.s	11
3.6	scsi.s	12
3.7	5380.h	14
3.8	68153.h	15
3.9	68230.h	15
3.10	68450.h	16
3.11	68881.h	16
3.12	8530.h	16
3.13	8536.h	16
3.14	gmsv04.h	16
3.15	gmsv06.h	16
3.16	hist.h	17
3.17	rt.h	17
3.18	scsi.h	17
3.19	tape.h	17
3.20	turbin.h	18
3.21	vme402.h	19
3.22	bdos.c	19
3.23	bios.c	23
3.24	change.c	23
3.25	cmdlx.c	24
3.26	disabl.c	24
3.27	enable.c	24
3.28	extern.c	25
3.29	getdat.c	25
3.30	getl.c	26
3.31	gettim.c	26
3.32	histin.c	27
3.33	histog.c	28
3.34	histpl.c	30
3.35	initpo.c	31
3.36	intl.c	32
3.37	putc.c	32
3.38	putl.c	32
3.39	query.c	32
3.40	queryi.c	33

3.41	queryl.c	33
3.42	querys.c	34
3.43	queryt.c	34
3.44	realti.c	35
3.45	rtinit.c	38
3.46	rtplot.c	39
3.47	scrnin.c	40
3.48	setdat.c	40
3.49	settim.c	40
3.50	setup.c	41
3.51	strlx.c	41
3.52	takeda.c	41
3.53	tapein.c	42
3.54	tapewr.c	42
3.55	title.c	44
3.56	turbin.c	44
3.57	twfm.c	45
3.58	update.c	46
3.59	writed.c	46
3.60	xylx.c	48
4.0	SLAVE PROGRAM SOFTWARE LISTINGS	49
4.1	a.s	49
4.2	hadcom.s	49
4.3	badexc.s	49
4.4	bladei.s	49
4.5	dispat.s	50
4.6	get1.s	50
4.7	get2.s	51
4.8	get3.s	51
4.9	get4.s	51
4.10	getn.s	51
4.11	histog.s	52
4.12	minmax.s	54
4.13	put1.s	55
4.14	putn.s	55
4.15	realti.s	56
4.16	receiv.s	57
4.17	reques.s	58
4.18	send.s	59
4.19	setup.s	59
4.20	slave.s	60
4.21	v04ini.s	62
4.22	v06ini.s	64
5.0	SIMULATOR EPROM LISTINGS	65

1.0 MAGNETIC TAPE FORMAT

NSMS Data Acquisition System - Tape Format

Fixed length - 512 byte records

block 0

bytes 0.. 3 : nb = number of blades
bytes 4.. 7 : nr = number of revolutions
bytes 8.. 11 : ns = number of stations
bytes 12.. 15 : average speed
bytes 16.. 19 : minimum speed
bytes 20.. 23 : maximum speed
bytes 24..127 : unused
bytes 128..136 : run date (DD-MMM-YY)
bytes 137..141 : excitation frequency
bytes 142..149 : run time (HH:MM:SS)
bytes 150..165 : radius
bytes 166..245 : run description
bytes 246..261 : run id
bytes 262..421 : specimen description
bytes 422..437 : specimen id
bytes 438..511 : unused

nbrd = number blocks for rev data = $\text{int}((nr + 127)/128)$

nbdd = number blocks for blade data = $\text{int}((nr*nb+127)/128)$

block 1..nbrd

bytes 0..3 : rev 1 time for station 1

bytes 4..7 : rev 2 time for station 1

.

.

block nbrd+1..nbrd+nbdd

bytes 0..3 : rev 1 blade 1 time for station 1

bytes 4..7 : rev 1 blade 2 time for station 1

.

.

bytes nb*4-4..nb*4-1 : rev 1 blade nb time for station 1

bytes nb*4..nb*4+1 : rev 2 blade 1 time for station 1 .

.

repeat from block 1 for station 2, 3 and 4

tape mark

2.0 COMPILATION CONTROL FILES

2.1 rom.com

```
*-----*  
run c:lnk  
-db0x1000 -eb_memory -ed_edata -et_etext -i -ovm:rom -tb0xf00000 <^1.l
```

2.2 call.com

```
*-----*  
c68k bdos  
c68k bios  
c68k change  
c68k cmd1x  
c68k disabl  
c68k enable  
c68k extern  
c68k getdat  
c68k getl  
c68k gettim  
c68k histin  
c68k histog  
c68k histpl  
c68k initpo  
c68k int1x  
c68k putc  
c68k putl  
c68k query  
c68k queryi  
c68k queryl  
c68k querys  
c68k queryt  
c68k realti  
c68k rtinit  
c68k rtplot  
c68k scrnin  
c68k setdat  
c68k settim  
c68k setup  
c68k strix  
c68k takeda  
c68k tapein  
c68k tapewr  
c68k title  
c68k turbin  
c68k twfm  
c68k update  
c68k writed  
c68k xy1x
```

2.3 aall.com

```
*-----*  
a68k clockr  
a68k clockw  
a68k cpm  
a68k revers  
a68k risr  
a68k scsi
```

2.4 turbin.l

```
*-----*  
vm:cpm.o  
vm:bdos.o  
vm:bios.o  
vm:change.o  
vm:clockr.o  
vm:clockw.o  
vm:cmd1x.o  
vm:disabl.o  
vm:enable.o  
vm:extern.o  
vm:getdat.o  
vm:getl.o
```



```
vm:gettinfo.o
vm:histinfo.o
vm:histlog.o
vm:histplot.o
vm:inittop.o
vm:inltx.o
vm:putc.o
vm:putl.o
vm:query.o
vm:queryrl.o
vm:queryrl.o
vm:queryrs.o
vm:queryrt.o
vm:realtime.o
vm:revers.o
vm:risc.o
vm:rtinit.o
vm:rtplot.o
vm:scrninfo.o
vm:scsi.o
vm:setdata.o
vm:settime.o
vm:setup.o
vm:strix.o
vm:takedata.o
vm:tapeinfo.o
vm:tapewr.o
vm:title.o
vm:turbinfo.o
vm:twf.o
vm:update.o
vm:writed.o
vm:xylx.o
c:lbpcpm.68k
c:lbpcpm.68k
```

2.5 rom.com

```
*-----*
run c:lnk
-db0x1000 -eb__memory -ed__edata -et__etext -i -tb0xf00000 -tf5 <^1.l
run c:hex
-m^1 -s >^1.com
```

2.6 aall.com

```

*-----*
run c:as68k a.s
run c:as68k bladei.s
run c:as68k badcom.s
run c:as68k badexc.s
run c:as68k dispat.s
run c:as68k get1.s
run c:as68k get2.s
run c:as68k get3.s
run c:as68k get4.s
run c:as68k getn.s
run c:as68k histog.s
run c:as68k minmax.s
run c:as68k put1.s
run c:as68k putn.s
run c:as68k reali.s
run c:as68k receiv.s
run c:as68k reques.s
run c:as68k send.s
run c:as68k setup.s
run c:as68k save.s
run c:as68k v04ini.s
run c:as68k v06ini.s

```

2.7 slave.1

```
*-----*
a.o
badcom.o
badexc.o
bladei.o
dispat.o
```

get1.o
get2.o
get3.o
get4.o
getn.o
histog.o
minmax.o
put1.o
putn.o
realti.o
receiv.o
reques.o
send.o
setup.o
slave.o
v04ini.o
v06ini.o

3.0 MASTER PROGRAM SOFTWARE LISTINGS

3.1 clockr.s

```
*-----*
.text

*   write = pc2
*   read  = pc1
*   hold  = pc0
*
*   a3:a0 = pb7:pb4
*   d3:d0 = pb3:pb0

clock= 0xfe0200
pgcr=  0+0+1
pbddr= 3+3+1
pcddr= 4+4+1
pbcr=  7+7+1
pbdr=  9+9+1
pbar= 11+11+1
pcdr= 12+12+1

HOLD= 0
READ= 1
WRITE= 2

.globl _clock_r
_clock_r:
    lea    clock,a0

    move.b #0x0.,pcddr(a0)

    move.b #0x01,pcdr(a0)

    move.b #0xf0,pbddr(a0)

*   delay 150 micro seconds
    move.w #150,d0
1:    dbf   d0,1b

    move.b #0x03,pcdr(a0)

    move.l 4(sp),a1

    clr.b d1

2:    move.b d1,pbdr(a0)

*   delay 6 micro seconds
    move.w #6,d0
3:    dbf   d0,3b

    move.b pbar(a0),d0

    and.b #0x000f,d0

    move.b d0,(a1)+

    add.b #0x10,d1

    cmp.b #0xc0,d1

    bls.s 2b

    move.b #0x00,pcdr(a0)

    rts
```

3.2 clockw.s

```
*-----*
.text

*   write = pc2
```

```

*      read = pc1
*      hold = pc0
*
*      a3:a0 = pb7:pb4
*      d3:d0 = pb3:pb0

clock= 0xfe0200
pgcr= 0+0+1
pbddr= 3+3+1
pcddr= 4+4+1
pbcr= 7+7+1
pbdr= 9+9+1
pbar= 11+11+1
pcdr= 12+12+1

HOLD= 0
READ= 1
WRITE= 2

        .globl _clock_w
_clock_w:
        lea    clock,a0

        move.b #0x07,pcddr(a0)

        move.b #0x01,pcdr(a0)

        move.b #0xff,pbddr(a0)

*      delay 150 micro seconds
1:      move.w #150,d0
        dbf    d0,1b

        move.l 4(sp),a1

        clr.b d1

2:      move.b (a1)+,d0

        and.b #0x000f,d0

        or.b d1,d0

        move.b d0,pbdr(a0)

        nop

        nop

        move.b #0x05,pcdr(a0)

        nop

        nop

        nop

        nop

        move.b #0x01,pcdr(a0)

        nop

        add.b #0x10,d1

        cmp.b #0xc0,d1

        bls.s 2b

        move.b #0x00,pcdr(a0)

        move.b #0x00,pbddr(a0)

        rts

```

3.3 cpm.s

```

*-----*
    .globl _edata
    .globl _ltor
    .globl _main
    .globl _memory
    .globl _rtol
    .globl _setint
    .globl _svc
    .globl _cpm
    .globl _exit

    .text

AmZ8536=0xfe0000
AmZ8530=0xfe0500
ac=    2+1
ad=    6+1
bc=    0+1
bd=    4+1

1:    .long 0x1000
      .long 2f

2:    move.l 1b,sp

*      clear memory - takes about 1 second
*      we can't do this in a subroutine because the return address would
*      be cleared also

      lea    0x000000,a0
      move.w #8-1,d1
      move.w #65536-8192-1,d0    * clear 2 M - 8 K
4:    clr.l  (a0)+
      dbf    d0,4b
      dbf    d1,4b

*      setup configuration port
      lea    AmZ8536,a0
      move.b 7(a0),d0
      nop
      nop
      move.b #0,7(a0)
      nop
      nop
      move.b 7(a0),d0
      lea    table,a1
      move.w #tablesize-1,d0
5:    move.b (a1)+,7(a0)
      dbf    d0,5b

*      copy data from prom to ram
*      get where data segment should be (above stack)
      move.l sp,a0
*      compute size of data segment
      move.l #_edata,d0
      sub.l  a0,d0
      beq    7f

*      get start of where data segment is (end of text segment)
      lea    _etext,a1
      subq.w #1,d0
6:    move.b (a1)+,(a0)+
      dbf    d0,6b

7:

*      setup all vectors to point to bad exception
      lea    0x000004,a0
      lea    bad_exception,a1
      move.w #256-1-1,d0
8:    move.l a1,(a0)+
      dbf    d0,8b

*      setup serial port
      lea    AmZ8530+ac,a0
      tst.b  (a0)    * make sure we will access R0 on next write
      nop           * 0.4 microseconds

```

```

nop          * 0.4 microseconds
nop          * 0.4 microseconds
nop          * 0.4 microseconds
move.b #0xc0,(a0) * reset 8530
lea sptable,a1 * 1.2 microseconds
move.w #sptsize-1,d1 * 0.8 microseconds
9: nop          * 0.4 microseconds
nop          * 0.4 microseconds
move.b (a1)+,(a0)
dbf d1,9b    * 1.0 microsecond

mtsr #0x2000
jsr _main
move.l d7,-(sp)
jsr _exit
trap #15
.word 0
bra 2b

__ltor:      move.l 4(sp),a0
move.l 8(sp),a1
add.l #1,a1
move.b (a1)+,(a0)+
move.b (a1)+,(a0)+
move.b (a1),(a0)
rts

__rtol:      move.l 4(sp),a0
move.l 8(sp),a1
move.b #0,(a0)+
move.b (a1)+,(a0)+
move.b (a1)+,(a0)+
move.b (a1),(a0)
rts

__setint:
rts

__svc:
_cpm: move.l 4(sp),d0
move.l 8(sp),d1
movem.l d2/d6/a0-a2,-(sp)
move.l d1,-(sp)
move.l d0,-(sp)
jsr _bdos
addq.l #8,sp
movem.l (sp)+,d2/d6/a0-a2
rts

bad_exception:
bset #3,AmZ8536+3 * turn on FAIL led
rte

sptable:.byte 4,0x44 * x16 clock, 1 stop bit, no parity
.byte 3,0xc0 * Rx=8 bits, Rx disabled
.byte 5,0x62 * Tx=8 bits, DTR=0, RTS=1, Tx disabled
.byte 9,0x00 * Int. Disabled
.byte 10,0x00 * NRZ
.byte 11,0x56 * RxC=BRG, TxC=BRG, TRxC Out=BRG
.byte 12,0x0b,15,0x00 * BRG Time Constant (9600 baud)
.byte 14,0x62 * Disable DPLL, BRG Source=PCLK, BRG disabled
.byte 14,0x63 * Disable DPLL, BRG Source=PCLK, BRG enabled
.byte 3,0xc1 * Rx=8 bits, Rx enabled
.byte 5,0x68 * Tx=8 bits, DTR=0, RTS=1, Tx enabled

sptsize=-sptable
table:.byte 0,1,0
.byte 0x05,0x00 * Port C Data Path Polarity
.byte 0x06,0x0e * Port C Data Direction
.byte 0x07,0x00 * Port C Special I/O Control
.byte 0x08,0x00 * Port A Command and Status
.byte 0x09,0x00 * Port B Command and Status
.byte 0x20,0x00 * Port A Mode Specification
.byte 0x21,0x00 * Port A Handshake Specification
.byte 0x22,0x00 * Port A Data Path Polarity
.byte 0x23,0x08 * Port A Data Direction
.byte 0x24,0x00 * Port A Special I/O Control
.byte 0x28,0x00 * Port B Mode Specification

```

```

.byte 0x29,0x00 * Port 3 Handshake Specification
.byte 0x2a,0x00 * Port B Data Path Polarity
.byte 0x2b,0xb0 * Port B Data Direction
.byte 0x2c,0x00 * Port B Special I/O Control
.byte 0x1e,0x95 * Counter/Timer 3 Mode Specification
.byte 0x0d,0xee * Port A Data
PA7 : NMIE* = 1
* PA6 : RAMCO = 1
* PA5 : MAPRO* = 1
* PA4 : WAITO = 0
* PA3 : PROM3 = 1
* PA2 : PROM2 = 1
* PA1 : PROM1 = 1
* PA0 : PROM0 = 0
.byte 0x0e,0x47 * Port B Data
PB7 : HALT* = ?
* PB6 : RESDIS* = 1
* PB5 : RESERVED = ?
* PB4 : SYSFAIL* = ?
* PB3 : FAIL = 0
* PB2 : RELES = 1
* PB1 : BUSL1 = 1
* PB0 : BUSLO = 1
.byte 0x0f,0xee * Port C Data
PC3 : CONTL = ?
* PC2 : DS = ?
* PC1 : CONTRL = ?
* PC0 : TIMEOUT = 1
.byte 0x01,0x94

```

tablesize= .-table

3.4 revers.s

```

*-----*
.text
.even
_reverse:
    movem.l    d0-d1/a0,-(sp)
    move.l 16(sp),a0
    move.l 20(sp),d1
    tst.w d1
    beq 2f
    subq.w #1,d1
1:    move.l (a0),d0
    rol.w #8,d0
    swap d0
    rol.w #8,d0
    move.l d0,(a0)+
    dbf d1,1b
2:    movem.l    (sp)+,d0-d1/a0
    rts
.globl _reverse

```

3.5 risr.s

```

*-----*
*
*    r_isr
*
*    This interrupt service routine handles the rev interrupts
*
*    At 6000 RPM (100 RPS) rev interrupts will occur
*    approximately every 9936 microseconds.
*
gmsv04= 0xffff000
port2= 0x0041
port3= 0x0081
debug= 0x0101

pbcr= 0x0e
padr= 0x10
pbdrr= 0x12
pcdr= 0x18

*    offsets in opstat
data_avail= 0
data_written= 1
taking_data= 2

```

```

.globl _rd_end,_rd_ptr
.globl _r_isr

_r_isr:
    movem.l    d0/a0-a1,-(sp)          * 32
    lea    gmv04,a0                    * 12
    move.b   port3+paddr(a0),d0        * 12
    andi.w   #0x000f,d0                * 8
    swap     d0                        * 4
    movep.w   port2+paddr(a0),d0        * 16
    move.l    _rd_ptr,a1                * 16
    move.l    d0,(a1)+                  * 8
    move.l    a1,_rd_ptr                * 16
    cmp.l     _rd_end,a1                * 22
    *         blo.s   1f                  * 10
    *         bcs.s   1f                  * 10
    *         clear interrupt enable
    *         bclr    #1,port2+paddr(a0)
    *         if we were taking data, tell program data is available
    *         move.b   _opstat+taking_data,_opstat+data_avail
    *         clr.b    _opstat+taking_data

1:     movem.l    (sp)+,d0/a0-a1        * 36
    rte                                     * 24
    *         ---
    *         216
    *         interrupt processing time 46
    *         ---
    *         total interrupt service time    262 cycles => 26.2 microseconds

```

3.6 scsi.s

```

*-----*
SCSI=      0xFE0300

Current_Data=      0+0+1
Output_Data=      0+0+1
Initiator_Command= 1+1+1
Mode=      2+2+1
Target_Command=    3+3+1
Current_Bus_Status= 4+4+1
Select_Enable=     4+4+1
Bus_and_Status=    5+5+1
Start_DMA_Send=    5+5+1
Input_Data=      6+6+1
Start_DMA_Target_Receive= 6+6+1
Reset_Parity_Interrupts= 7+7+1
Start_DMA_Initiator_Receive= 7+7+1

BSY~= 6
REQ~= 5
MSG~= 4
C_D~= 3
I_O~= 2

PHASE_MATCH= 3

ASSERT_ACK~= 4
ASSERT_SEL~= 2
ASSERT_DATA_BUS=0

.globl _scsi_in

*
*     scsi_in(phase, buffer, length);
*
_scsi_in:
    lea     SCSI,a0
    *
    *     4(sp) = phase
    *     8(sp) -> buffer
    *     12(sp) = length
    *
    *     copy arguments to registers
    *
    move.b  7(sp),d0

```



```

    move.l 8(sp),a1
    move.l 12(sp),d1

    move.b d0,Target_Command(a0)

    clr.b d0

1:   btst  #PHASE_MATCH,Bus_and_Status(a0)
    beq.s 1b

2:   btst  #REQ~,Current_Bus_Status(a0)
    beq.s 2b

    move.b Current_Data(a0),(a1)+

    bset  #ASSERT_ACK~,d0
    move.b d0,Initiator_Command(a0)

3:   btst  #REQ~,Current_Bus_Status(a0)
    bne.s 3b

    bclr  #ASSERT_ACK~,d0
    move.b d0,Initiator_Command(a0)

    subq.l #1,d1

    bne.s 1b

    rts

.globl _scsi_out
*
*   scsi_out(phase, buffer, length);
*
_scsi_out:
    lea    SCSI,a0
*
*   4(sp) = phase
*   8(sp) -> buffer
*   12(sp) = length
*
*   copy arguments to registers
*
    move.b 7(sp),d0
    move.l 8(sp),a1
    move.l 12(sp),d1

    move.b d0,Target_Command(a0)

    clr.b d0
    bset  #ASSERT_DATA_BUS,d0
    move.b d0,Initiator_Command(a0)

1:   btst  #PHASE_MATCH,Bus_and_Status(a0)
    beq.s 1b

2:   btst  #REQ~,Current_Bus_Status(a0)
    beq.s 2b

    move.b (a1)+,Current_Data(a0)

    bset  #ASSERT_ACK~,d0
    move.b d0,Initiator_Command(a0)

3:   btst  #REQ~,Current_Bus_Status(a0)
    bne.s 3b

    bclr  #ASSERT_ACK~,d0
    move.b d0,Initiator_Command(a0)

    subq.l #1,d1

    bne.s 1b

    bclr  #ASSERT_DATA_BUS,d0
    move.b d0,Initiator_Command(a0)

```

```

        rts
        .globl _scsi_sel
*
*       scsi_sel(device)
*
_scsi_sel:
        lea     SCSI,a0

1:       move.w 0xffff,d7
        bstat  #BSY~,Current_Bus_Status(a0)
        dbeq   d7,1b

        cmp.w  #-1,d7
        beq.e  3f

        clr.b  Target_Command(a0)

        move.b 7(sp),Output_Data(a0)

        clr.b  d0

        bset   #ASSERT_DATA_BUS,d0
        move.b d0,Initiator_Command(a0)

        bset   #ASSERT_SEL~,d0
        move.b d0,Initiator_Command(a0)

2:       move.w #0xffff,d7
        bstat  #BSY~,Current_Bus_Status(a0)
        dbne   d7,2b

        bclr   #ASSERT_SEL~,d0
        move.b d0,Initiator_Command(a0)

        bclr   #ASSERT_DATA_BUS,d0
        move.b d0,Initiator_Command(a0)

3:       addq.l #1,d7
        beq.s  4f

4:       moveq.l    #1,d7
        rts

```

3.7 5380.h

```

*-----*
struct NCR5380 {
    unsigned char 00;
    unsigned char CurrentData;
    unsigned char 02;
    unsigned char InitiatorCommand;
    unsigned char 04;
    unsigned char Mode;
    unsigned char 06;
    unsigned char TargetCommand;
    unsigned char 08;
    unsigned char CurrentBusStatus;
    unsigned char 0a;
    unsigned char 0b;
    unsigned char 0c;
    unsigned char 0d;
    unsigned char 0e;
    unsigned char 0f;
};

/* current bus status register */
#define SEL 0x02

/* mode register */
#define TARGET_MODE 0x40

/* target command register */
#define DATA_OUT 0
#define DATA_IN 1

```

```
#define COMMAND 2
#define STATUS 3
#define MESSAGE_IN 7
```

3.8 68153.h

```
*-----*
struct MC68153 {
    unsigned char _00;
    unsigned char cr_0;
    unsigned char _02;
    unsigned char cr_1;
    unsigned char _04;
    unsigned char cr_2;
    unsigned char _06;
    unsigned char cr_3;
    unsigned char _08;
    unsigned char vr_0;
    unsigned char _0a;
    unsigned char vr_1;
    unsigned char _0c;
    unsigned char vr_2;
    unsigned char _0e;
    unsigned char vr_3;
};
```

3.9 68230.h

```
*-----*
#ifndef _MC68230
#define _MC68230
struct MC68230 {
    unsigned char _00;
    unsigned char pocr;
    unsigned char _02;
    unsigned char perr;
    unsigned char _04;
    unsigned char paddr;
    unsigned char _06;
    unsigned char pbaddr;
    unsigned char _08;
    unsigned char pcaddr;
    unsigned char _0a;
    unsigned char pivr;
    unsigned char _0c;
    unsigned char pacr;
    unsigned char _0e;
    unsigned char pocr;
    unsigned char _10;
    unsigned char padr;
    unsigned char _12;
    unsigned char pbdr;
    unsigned char _14;
    unsigned char pear;
    unsigned char _16;
    unsigned char pbar;
    unsigned char _18;
    unsigned char pcdr;
    unsigned char _1a;
    unsigned char par;
    unsigned char _1c[5];
    unsigned char tcr;
    unsigned char _22;
    unsigned char tivr;
    unsigned char _24[3];
    unsigned char cpr_h;
    unsigned char _28;
    unsigned char cpr_m;
    unsigned char _2a;
    unsigned char cpr_l;
    unsigned char _2c[3];
    unsigned char cr_h;
    unsigned char _30;
    unsigned char cr_m;
    unsigned char _32;
};
```

```

        unsigned char cr_l;
        unsigned char _34;
        unsigned char tsr;
        unsigned char _36[10];
};

```

```

#endif

```

3.10 68450.h

```

*-----*
struct MC68450 {
    unsigned char _00[0x40];
};

```

3.11 68881.h

```

*-----*
#ifndef _MC68881
#define _MC68881

struct MC68881 {
    unsigned short _0[8];
};

#endif

```

3.12 8530.h

```

*-----*
struct Z8530 {
    unsigned char _00;
    unsigned char b_control;
    unsigned char _02;
    unsigned char a_control;
    unsigned char _04;
    unsigned char b_data;
    unsigned char _06;
    unsigned char a_data;
};

```

3.13 8536.h

```

*-----*
struct Z8536 {
    unsigned char _00;
    unsigned char a_data;
    unsigned char _02;
    unsigned char b_data;
    unsigned char _04;
    unsigned char c_data;
    unsigned char _06;
    unsigned char control;
};

```

3.14 gmsv04.h

```

*-----*
#include "68153.h"

#include "68230.h"

struct GMSV04 {
    unsigned short port_1[32];
    struct MC68230 port_2;
    struct MC68230 port_3;
    struct MC68230 config;
    struct MC68230 debug;
    struct MC68153 intsel;
};

```

3.15 gmsv06.h

```

*-----*
#include "5380.h"

#include "68153.h"

```

```

#include "68230.h"
#include "68450.h"
#include "68881.h"
#include "8530.h"
#include "8536.h"

struct GMSV06 {
    struct Z8536 config;
    unsigned char _008[0xf8];
    struct MC68153 int_cntrlr;
    unsigned char _110[0xf0];
    struct MC68230 rtc_pp;
    unsigned char _240[0xc0];
    struct MCR5380 scsi;
    unsigned char _310[0xf0];
    struct MC68450 dma;
    unsigned char _440[0xc0];
    struct Z8530 serial;
    unsigned char _508[0xf8];
    struct MC68881 fpcp;
};

#define gmsv06 (*((struct GMSV06 *)0xfe0000))

```

3.16 hist.h

```

*-----*
#define XOFFSET 60
#define XSIZE 420
#define YOFFSET 8
#define YSIZE 300

```

3.17 rt.h

```

*-----*
#define MAX_RT_BLADES 5
#define MAX_RT_REVS 10

#define XOFFSET 79
#define XSIZE 400
#define YOFFSET 8
#define YSIZE 300

```

3.18 scsi.h

```

*-----*
/* scsi commands */
#define HISTOGRAM 0x02
#define REQUEST_SENSE 0x03
#define REAL_TIME_DISPLAY 0x06
#define RECEIVE 0x08
#define SETUP 0x09
#define SEND 0x0a

/* scsi phases */
#define DATA_OUT 0
#define DATA_IN 1
#define COMMAND 2
#define STATUS 3
#define MESSAGE_OUT 6
#define MESSAGE_IN 7

```

3.19 tape.h

```

*-----*
#define BASE 0xffff00

#define DMA_DATA ((short *) (BASE + 0x10))
#define DMA_ADDR ((short *) (BASE + 0x12))

#define COMMAND ((unsigned char *) (BASE + 0x1b))
#define CONTROL ((unsigned char *) (BASE + 0x19))
#define DATA ((unsigned char *) (BASE + 0x1d))

```

```

#define STATUS_0 ((unsigned char *) (BASE + 0x19))
#define STATUS_1 ((unsigned char *) (BASE + 0x1b))

#define FMK 0x01 /* File Mark detected */
#define EOT 0x02 /* End Of Tape detected */
#define HER 0x04 /* Hard Read Error */
#define CER 0x08 /* Corrected Read Error */
#define DLOST 0x10 /* No data lost */
#define TDREQ 0x20 /* No data byte requested */
#define FBY 0x40 /* Formatter not busy */
#define DBY 0x80 /* Not in Data Transfer Mode */

#define SPD 0x01
#define NRZ 0x02
#define CCG 0x04
#define LPT 0x08
#define FPT 0x10
#define RWD 0x20
#define ONL 0x40
#define RDY 0x80

#define CCEN 0x01
#define LOL 0x02
#define REW 0x04
#define OFL 0x08
#define FEN 0x10
#define FAD 0x20
#define TAD 0x40

#define REV 0x01
#define ERASE 0x02
#define EDIT 0x04
#define THR1 0x08
#define THR2 0x10
#define DEN 0x20
#define WFM 0x40
#define WRT 0x80

```

3.20 turbin.h

```

*-----*
#define isdigit(x) ((x) >= '0' && (x) <= '9')
#define void int
#define TRUE 1
#define FALSE 0

#define SLAVE_BSS 740
#define SLAVE_DATA 0
#define SLAVE_END 0x200000
#define SLAVE_MEMSIZE (SLAVE_END-0x1000-SLAVE_DATA-SLAVE_BSS)
#define MAX_BLADES 70
#define MAX_STATIONS 4
#define TWO7 140737488355328.0

#define VECTOR 64

struct HEADER {
    long n_blades;
    long n_revs;
    long n_stations;
    long ave_speed;
    long min_speed;
    long max_speed;
    long _00c(26);
    char date(9);
    char excit_freq(5);
    char time(8);
    char radius(16);
    char run_des(80);
    char run_id(16);
    char sp_des(2)(80);
    char sp_id(16);
    char _1b6(74);
};

/* NOTE: if any elements of structure OPSTAT are changed, the routine
risc.s must be changed also */

```

```

struct OPSTAT (
    char data_avail;
    char data_written;
    char taking_data;
);

```

3.21 vme402.h

```

*-----*
#define BASE 0xffff00

/* DMA definitions */

#define DMA_ADDR ((short *) (BASE + 0x12))
#define DMA_DATA ((short *) (BASE + 0x10))

#define MMR      0x38
#define CR       0x2e
#define CAR_A_HI 0x1a
#define CAR_A_LO 0x0a
#define CAR_B_HI 0x12
#define CAR_B_LO 0x02
#define COC      0x32
#define CMR_HI   0x56
#define CMR_LO   0x52
#define CHAR_HI  0x26
#define CHAR_LO  0x22

/* tape definitions */

#define TAPE_COMMAND_STATUS_1 ((unsigned char *) (BASE + 0x1b))
#define TAPE_CONTROL_STATUS_0 ((unsigned char *) (BASE + 0x19))
#define TAPE_DATA              ((unsigned char *) (BASE + 0x1d))

#define FNK 0x01 /* File Mark detected */
#define EOT 0x02 /* End Of Tape detected */
#define HER 0x04 /* Hard Read Error */
#define CER 0x08 /* Corrected Read Error */
#define DLOST 0x10 /* No data lost */
#define TDREQ 0x20 /* No data byte requested */
#define FBY 0x40 /* Formatter not busy */
#define DBY 0x80 /* Not in Data Transfer Mode */

#define SPD 0x01
#define NRZ 0x02
#define CCG 0x04
#define LPT 0x08
#define FPT 0x10
#define RWD 0x20
#define ONL 0x40
#define RDY 0x80

#define CCEN 0x01
#define LOI 0x02
#define REW 0x04
#define OFL 0x08
#define FEN 0x10
#define FAD 0x20
#define TAD 0x40

#define REV 0x01
#define ERASE 0x02
#define EDIT 0x04
#define THR1 0x08
#define THR2 0x10
#define DEN 0x20
#define WFM 0x40
#define WRT 0x80

#define READ 0xff
#define WRITE 0x7f
#define T_WFM 0x3f

```

3.22 bdos.c

```

*-----*
/*****

```

```

/*
/* bdos.c
/*
/* This file implements (except for handling ^C) the following bdos calls :
/*
/* 1 - Console Input
/* 2 - Console Output
/* 10 - Read Console Buffer
/* 11 - Console Status
/*
/*
/*****

short column = 0;
char echo_to_list = 0;
char crt = 0;
char ta_buffer[126] = 0;
char ta_n_char = 0;
char *ta_get_ptr = ta_buffer;
char *ta_put_ptr = ta_buffer;

bdos(d0, d1)
short d0;
long d1;
{
    auto char string[81];

    if (d0 == 1) {
        /* Console Input */
        d0 = con_input();
    } else if (d0 == 2) {
        /* Console Output */
        con_output(d1);
        d0 = 0;
    } else if (d0 == 10) {
        /* Read Console Buffer */
        read_console_buffer(d1);
        d0 = 0;
    } else if (d0 == 11) {
        /* Get Console Status */
        d0 = con_status();
    } else {
        sprintf(string, "Error - bdos function %d not implemented", d0);
        print(string);
        d0 = -1;
    }
    return d0;
}

/* $38cea */
con_input()
{
    register char c;
    put_char(c = get_char());
    if (c == '\020')
        echo_to_list = ! echo_to_list;
    return c;
}

/* $38c12 */
con_output(c)
register char c;
{
    if (c == '\t') {
        do {
            put_char(' ');
        } while ((column & 0x07) != 0);
    } else {
        put_char(c);
    }
}

/* $38af2 */
con_status()
{
    if (ta_n_char != 0)
        return (1);
    else

```



```

        return (bios(2));
    }

/* $38e46 */
read_console_buffer(buffer)
register char buffer[];
{
    register char c;
    register short i;
    short old_column;

    old_column = column;
    buffer[1] = 0;
    while (buffer[1] < buffer[0]) {
        if ((c = get_char()) == 3 && buffer[1] == 0)
            ;
        if (c == '\r' || c == '\n') {
            put_char('\r');
            return;
        } else if (c == '\b') {
            backspace(buffer, old_column);
        } else if (c == '\177') {
            if (crt != 0)
                backspace(buffer, old_column);
            else
                if (buffer[1] != 0) {
                    --buffer[1];
                    put_char(buffer[buffer[1] + 2]);
                }
        } else if (c == '\020') {
            echo_to_list = ! echo_to_list;
        } else if (c == '\030') {
            /* control-X (CAN) */
            do
                backspace(buffer, old_column);
            while (buffer[1] != 0);
        } else if (c == '\005') {
            /* control-E */
            newline(old_column);
        } else if (c == '\025') {
            /* control-U */
            put_char('#');
            newline(old_column);
            buffer[1] = 0;
        } else if (c == '\022') {
            /* control-R */
            put_char('#');
            newline(old_column);
            for (i = 0; i < buffer[1]; i++)
                echo(buffer[i+2]);
        } else {
            buffer[buffer[1] + 2] = c;
            echo(c);
            buffer[1]++;
        }
    }
}

/* $38c50 */
echo(c)
register char c;
{
    if (c == '\t') {
        con_output(c);
    } else if (c < ' ') {
        put_char('^');
        put_char(c | 0x40);
    } else {
        put_char(c);
    }
}

/* $38c92 */
get_char()
{
    register char c;

```

```

        if (ta_n_char != 0) {
            c = *ta_get_ptr++;
            ta_n_char--;
            if (ta_n_char == 0) {
                ta_get_ptr = ta_buffer;
                ta_put_ptr = ta_buffer;
            }
            return (c);
        } else
            return (bios(3));
    }

/* $38d96 */
newline(column)
register short column;
{
    put_char('\r');
    put_char('\n');
    for ( ; column != 0 ; column--)
        put_char(' ');
}

/* $38dcc */
backspace(buffer, my_column)
register char buffer[];
register short my_column;
{
    register char c;
    register short count;
    register char *ptr;

    if (buffer[1] != 0)
        buffer[1]--;
    count = buffer[1];
    for (ptr = &buffer[2] ; count-- != 0 ; ) {
        c = *ptr++;
        if (c == '\t')
            my_column = (my_column + 8) & ~7;
        else if (c < ' ')
            my_column += 2;
        else
            my_column ++;
    }
    while (my_column < column) {
        put_char('\b');
        put_char(' ');
        put_char('\b');
    }
}

/* $38ba4 */
put_char(c)
register unsigned char c;
{
    check_xon();
    bios(4, c);
    if (echo_to_list)
        bios(5, c);
    if (c >= ' ')
        column++;
    else if (c == '\r')
        column = 0;
    else if (c == '\b')
        column--;
    return;
}

/* $38b10 */
check_xon()
{
    register char c;
    register char flag;

    flag = 0;
    do {
        if (bios(2) != 0) {
            if ((c = bios(3)) == 3) {

```

```

        /* control-C */
        ;
    } else if (c == 0x13) {
        /* control-S (XOFF) */
        flag = 1;
    } else if (c == 0x11) {
        /* control-Q (XON) */
        flag = 0;
    } else if (c == 0x10) {
        /* control-P */
        echo_to_list = ! echo_to_list;
    } else if (ta_n_char < sizeof(ta_buffer)) {
        *ta_put_ptr++ = c;
        ta_n_char++;
    }
}
} while (flag != 0);
}

```

```

print(strptr)
char *strptr;
{
    while (*strptr != '\0')
        bios(4, *strptr++);
}

```

3.23 bios.c

```

-----*
/*-----*/
/*
/* bios.c
/*
/* This file implements the following bios calls:
/*
/*
/* 2 - Console Status
/* 3 - Read Console Character
/* 4 - Write Console Character
/*
/*-----*/

#include "gmsv06.h"

#define Rx_CHARACTER_AVAILABLE 0x01
#define Tx_BUFFER_EMPTY 0x04

bios(d0, d1)
short d0;
short d1;
{
    if (d0 == 2) {
        /* Console Status */
        if ((gmsv06.serial.a_control & Rx_CHARACTER_AVAILABLE) != 0)
            d0 = 0xff;
        else
            d0 = 0;
    } else if (d0 == 3) {
        /* Read Console Character */
        while ((gmsv06.serial.a_control & Rx_CHARACTER_AVAILABLE) == 0)
            ;
        d0 = gmsv06.serial.a_data & 0x7f;
    } else if (d0 == 4) {
        /* Write Console Character */
        while ((gmsv06.serial.a_control & Tx_BUFFER_EMPTY) == 0)
            ;
        gmsv06.serial.a_data = d1;
        d0 = 0;
    } else {
        d0 = -1;
    }
    return d0;
}

```

3.24 change.c

```

-----*
#include <stdio.h>

```

```

#include "turbin.h"

change()
{
    auto    char    buffer[81];
    auto    char    date[10];
    extern  struct  HEADER header;
    extern  int     n_blades;
    extern  int     n_stations;
    auto    char    time[9];

    printf("\033[2J\033[1;1H"); /* Clear Screen and home */

    get_date(date);
    printf("The date is currently %s\n", date);
    if (query("Do you want to change the date"))
        set_date(date);

    get_time(time);
    printf("The time is currently %s\n", time);
    if (query("Do you want to change the time"))
        set_time(time);

    queryi("Number of stations", &n_stations, "", 0, MAX_STATIONS);

    querys("Specimen I.D.", header.sp_id, sizeof(header.sp_id), "");
    queryt("Specimen description", header.sp_des, sizeof(header.sp_des));
    querys("Specimen radius", header.radius, sizeof(header.radius), "in.");
    queryi("Number of blades", &n_blades, "", 1, 70);

    header.n_blades = n_blades;
    header.n_stations = n_stations;
}

```

3.25 cmd1x.c

```

*-----*
#include <stdio.h>

#define putchar(c) fputc(c, stdout);

cmd1x(opcode)
char *opcode;
{
    putchar('\033');
    putchar(*opcode);
    opcode++;
    putchar(*opcode);
}

```

3.26 disabl.c

```

*-----*
#include "gmsv04.h"

disable()
{
    register struct GMSV04 *gmsv04;

    gmsv04 = 0xfff000;

    /* deassert port_3's H4 to disable remote electronics */
    gmsv04 -> port_3.pbcr = 0x20;

    /* deassert port_2's H4 to clear data available flip-flop */
    gmsv04 -> port_2.pbcr = 0x20;
}

```

3.27 enable.c

```

*-----*
#include "gmsv04.h"

enable()

```

```

(
    register struct GMSV04 *gmsv04;
    register char          temp;

    gmsv04 = 0xfff000;

    /* clear buffers */
    while ((gmsv04 -> port_3.per & 0x04) != 0)
        temp = gmsv04 -> port_3.pbdr;
    while ((gmsv04 -> port_2.per & 0x04) != 0) {
        temp = gmsv04 -> port_2.padr;
        temp = gmsv04 -> port_2.pbdr;
    }

    /* set port_2's H4 to pulsed input handshake protocol */
    gmsv04 -> port_2.pbcr = 0x3a;

    /* assert port_3's H4 to enable remote electronics */
    gmsv04 -> port_3.pbcr = 0x28;
)

```

3.28 extern.c

```

*-----*
#include "turbin.h"

extern struct HEADER header = 0;
extern struct OPSTAT opstat = 0;

/* globals for interrupt service routines */
extern long *rd_end = 0;
extern long *rd_ptr = 0;
extern long *rd_start = &_memory;

/* globals for histogram routines */
extern int hist_n_bars = 0;
extern int hist_xstep = 0;
extern int hist_yold[MAX_BLADES] = 0;
extern int hist_yoffset = 0;

/* globals for tape routines */
extern char control = 0;
extern char last_op = 0;

extern int n_blades = 5;
extern int n_stations = 4;

extern long rtd_n_revs = 0;
extern int rtd_height = 0;
extern int rtd_n_blades = 0;
extern int rtd_xstep = 0;
extern int rtd_ystep = 0;
extern int rtd_ystart = 0;
extern int rtd_hd2 = 0;

```

3.29 getdat.c

```

*-----*
get_date(date)
register char *date;
(
    unsigned char buffer[13];
    register int i;
    static char months[13][3] = {
        {'J', '?', 'n'},
        {'J', 'a', 'n'},
        {'F', 'e', 'b'},
        {'M', 'a', 'r'},
        {'A', 'p', 'r'},
        {'M', 'a', 'y'},
        {'J', 'u', 'n'},
        {'J', 'u', 'l'},
        {'A', 'u', 'g'},
        {'S', 'e', 'p'},

```

```

        'O', 'c', 't',
        'N', 'o', 'v',
        'D', 'e', 'c'
    );

    register char *ptr;

    clock_r(buffer);
    i = buffer[8] & 0x03;
    if (i == 0)
        *date++ = ' ';
    else
        *date++ = i + '0';
    *date++ = buffer[7] + '0';
    *date++ = '-';
    i = buffer[10] & 0x01;
    if (i != 0)
        i = 10;
    i += buffer[9];
    if (i > 12)
        i = 0;
    ptr = &months[i][0];
    *date++ = *ptr++;
    *date++ = *ptr++;
    *date++ = *ptr++;
    *date++ = '-';
    *date++ = buffer[12] + '0';
    *date++ = buffer[11] + '0';
    *date++ = '\0';
}

```

3.30 get1.c

```

*-----*
unsigned int get1(pfio, s, n)
char *pfio;
char *s;
unsigned int n;
{
    auto unsigned char buffer[257];
    register unsigned int i;

    buffer[0] = n;

    cpm(10, buffer);

    i = buffer[1];

    if (i != n) {
        buffer[i+2] = '\n';
        cpm(2, '\n');
        i++;
    }

    n = i;

    for (i = 0; i < n; i++)
        *s++ = buffer[i + 2];

    return n;
}

```

3.31 gettim.c

```

*-----*
get_time(time)
register char *time;
{
    unsigned char buffer[13];

    clock_r(buffer);
    *time++ = (buffer[5] & 0x03) + '0';
    *time++ = buffer[4] + '0';
    *time++ = ':';
    *time++ = buffer[3] + '0';
    *time++ = buffer[2] + '0';
    *time++ = ':';
}

```

```

*time++ = buffer[1] + '0';
*time++ = buffer[0] + '0';
*time++ = '\0';

```

```

)

```

3.32 histin.c

```

*-----*
/*****
/*
/* hist_init
/* This routine sets up and labels the screen for a histogram
/*
/*****

#include <stdio.h>
#include "hist.h"

hist_init(n_bars)
int n_bars;
{
    extern int hist_n_bars;
    extern int hist_xstep;
    extern int hist_yoffset;
    extern int hist_yold[];
    int i;
    int n;
    char string[2];
    int x;

    hist_n_bars = n_bars;
    hist_xstep = XSIZE / n_bars;

    /* Determine # of lines at bottom of screen */

    if (hist_xstep < 12)
        hist_yoffset = YOFFSET + 16;
    else
        hist_yoffset = YOFFSET + 8;

    x = (hist_xstep >> 1) - 3 + XOFFSET;
    for (i = 0 ; i < hist_n_bars ; i++) {
        hist_yold[i] = hist_yoffset;
        n = i+1;
        string[0] = n / 10 + '0';
        if (string[0] == '0')
            string[0] = ' ';
        string[1] = n % 10 + '0';
        if (hist_xstep >= 12) {
            if (string[0] == ' ') {
                cmd1x("LF");
                xy1x(x, YOFFSET);
                cmd1x("LT");
                int1x(1);
                putchar(string[1]);
            } else {
                cmd1x("LF");
                xy1x(x - 3, YOFFSET);
                cmd1x("LT");
                int1x(2);
                putchar(string[0]);
                putchar(string[1]);
            }
        }
        cmd1x("LF");
        xy1x(x, YOFFSET + 8);
        cmd1x("LT");
        int1x(1);
        putchar(string[0]);
        cmd1x("LF");
        xy1x(x, YOFFSET);
        cmd1x("LT");
        int1x(1);
        putchar(string[1]);
    }
    x += hist_xstep;
}

```

)

3.33 histog.c

```

*-----*
/*****.******/
/*
/* histogram
/*
/* This routine performs the histogram quick look
/*
/* While this routine is running the following keys are recognized:
/*
/* ' ' - resets the full scale value to the current maximum
/*
/* '\r' - exits histogram mode after current histogram
/*
/*
/*****.******/

```

```

#include <stdio.h>
#include "turbin.h"
#include "scsi.h"

```

histogram()

```

(
  extern struct HEADER header; /* must be before sizeof(header.*) */
  extern struct OPSTAT opstat;
  auto int c;
  auto char cdb[6];
  register long *d_ptr;
  auto char date[10];
  static char excit_freq[sizeof(header.excit_freq) + 1];
  auto int exit;
  auto long first;
  auto float h_time;
  register int i;
  auto int id;
  auto char message_in;
  extern int n_blades;
  static int n_h_revs;
  auto long n_revs;
  auto long range[MAX_BLADES];
  static long rangemax;
  extern long *rd_end;
  extern long *rd_ptr;
  extern long *rd_start;
  auto long recip;
  auto long revs_left;
  static char run_id[sizeof(header.run_id) + 1];
  auto int speed;
  static int start_rev;
  static int station;
  auto char status;
  auto long sum;
  auto char time[9];

  if (opstat.data_avail && opstat.data_written)
    opstat.data_avail = FALSE;

  printf("\033[2J\033[1;1H"); /* Clear Screen and Home */

  queryi("Station for histogram", &station, "", 1, 4);
  id = 1 << (station - 1);

/*
  queryf("Time between histograms", h_time, "seconds", 0.5, 10.0);
  n_h_revs = h_time * (speed / 60.0);
*/
  queryi("Number of revs for histogram", &n_h_revs, "", 1, 999);

  if (opstat.taking_data) {
    for (i = sizeof(header.run_id) - 1; i >= 0; i--)
      if (header.run_id[i] != ' ')
        break;
    run_id[i+1] = '\0';
    while (i >= 0) {
      run_id[i] = header.run_id[i];
      i--;
    }
  }

```



```

    }

    for (i = sizeof(header.excit_freq) - 1 ; i >= 0 ; i--)
        if (header.excit_freq[i] != ' ')
            break;
    excit_freq[i+1] = '\0';
    while (i >= 0) {
        excit_freq[i] = header.excit_freq[i];
        i--;
    }
} else {
    opstat.data_avail = FALSE;
    querys("Histogram I.D.", run_id, -sizeof(run_id), "");
    querys("Excitation frequency", excit_freq, -sizeof(excit_freq), "");
}

scrn_init();

title(run_id, station, excit_freq);

hist_init(n_blades); /* label histogram screen */

rangemax = 0;
exit = FALSE;
do {
    get_date(date);
    get_time(time);
    if (opstat.taking_data) {
        start_rev = rd_ptr - rd_start;
        revs_left = rd_end - rd_ptr;
        if (revs_left < (n_h_revs + 2)) /* if we are finishing up */
            break; /* disallow histograms */
    } else if (opstat.data_avail) {
        /* we finished - exit */
        break;
    } else {
        disable();
        rd_ptr = rd_start;
        n_revs = n_h_revs + 3; /* since first rev of data is bad */
        /* and we need previos rev for fixup */
        rd_end = rd_start + n_revs;
        setup(station, n_revs);
        enable();
        start_rev = 3; /* since first rev of data is bad */
        /* and we need previos rev for fixup */
    }

    if (scsi_sel(id)) {
        cdb[0] = HISTOGRAM;
        cdb[1] = start_rev >> 16;
        cdb[2] = start_rev >> 8;
        cdb[3] = start_rev;
        cdb[4] = n_h_revs >> 8;
        cdb[5] = n_h_revs;
        scsi_out(COMMAND, cdb, 6);
        /* send previous rev time for back jitter of first blade */
        d_ptr = rd_start + (start_rev - 1);
        while (d_ptr >= rd_ptr)
            ;
        scsi_out(DATA_OUT, d_ptr++, 4);
        for (i = 0 ; i < n_h_revs ; i++) {
            while (d_ptr >= rd_ptr)
                ;
            if (i == 0) {
                first = *d_ptr;
                sum = 0;
            } else {
                sum += *d_ptr - first;
            }
            scsi_out(DATA_OUT, d_ptr, 4);
            recip = TWO47 / *d_ptr++ + 0.5;
            scsi_out(DATA_OUT, &recip, 4);
        }
        for (i = 0 ; i < n_blades , i++) {
            scsi_in(DATA_IN, &range[i], 4);
            if (range[i] > rangemax)
                rangemax = range[i];
        }
    }
}

```

```

    scsi_in(STATUS, &status, 1);
    scsi_in(MESSAGE_IN, &message_in, 1);
    if (status == 0) {
        speed = (60 * 1000000) / (first + (sum / n_h_revs));
        update(speed, date, time);
        hist_plot(range, rangemax);
    } else {
        printf("Error - data station had an error\n");
    }
} else {
    printf("Error - data station does not respond\n");
}

/* check if char available */
while (cpm(1, 0) != 0) {
    /* get char */
    c = cpm(1, 0);
    cmd1x("LZ"); /* CLEAR DIALOG SCROLL */
    if (c == '\r')
        rangemax = 0;
    else if (c == '\r' || c == '\n')
        exit = TRUE;
}
while (! exit);

printf("\033\014"); /* Clear graphics screen */
printf("\033XX11"); /* Select ANSI mode */
}

```

3.34 histpl.c

```

*-----*
/*****
*/
/* hist_plot
*/
/* This routine plots the histogram bars on the screen
*/
/*****

#include <stdio.h>
#include "hist.h"

hist_plot(data, datamax)
long data[];
long datamax;
{
    extern int hist_nBars;
    extern int hist_xstep;
    extern int hist_yoffset;
    extern int hist_yold[70];
    int i;
    int xl;
    int xr;
    int y;
    char string[11];

    xl = XOFFSET + 1;
    xr = XOFFSET + hist_xstep - 2;

    for (i = 0 ; i < hist_nBars ; i++) {
        y = (data[i] * YSIZE) / datamax + hist_yoffset;
        if (y < hist_yold[i]) {
            cmd1x("RR");
            xylx(xl, y + 1);
            xylx(xr, hist_yold[i]);
            int1x(0);
            hist_yold[i] = y;
        } else if (y > hist_yold[i]) {
            cmd1x("RR");
            xylx(xl, hist_yold[i] + 1);
            xylx(xr, y);
            int1x(1);
            hist_yold[i] = y;
        }
        xl += hist_xstep;
    }
}

```

```

        xr += hist_xstep;
    }

    /* Draw 0 level line */
    cmd1x("LF");
    xy1x(XOFFSET, hist_yoffset);
    cmd1x("LG");
    xy1x(XOFFSET+XSIZE, hist_yoffset);
    cmd1x("LF");
    xy1x(XOFFSET-10, hist_yoffset-3);
    cmd1x("LT");
    int1x(1);
    printf("0");

    /* Display Full Scale Maximum value & level */

    cmd1x("LF");
    xy1x(XOFFSET-58, hist_yoffset+YSIZE-3);
    cmd1x("LT");
    int1x(10);
    printf("%9d", datamax);

    cmd1x("LF");
    xy1x(XOFFSET, hist_yoffset+YSIZE+1);
    cmd1x("LG");
    xy1x(XOFFSET+XSIZE, hist_yoffset+YSIZE+1);
}

```

3.35 initpo.c

```

*-----*
#include "gmsv04.h"

init_port(vector)
register int vector;
{
    register struct GMSV04 *gmsv04;

    gmsv04 = 0xfff000;

    gmsv04 -> port_2.pgcr &= ~0x30; /* disable H12 and H34 */
    gmsv04 -> port_2.pgcr = 0x40; /* mode 1 (unidirectional 16-bit mode) */
    /* H34 disabled */
    /* H12 disabled */
    /* H4 pin sense - negative true */
    /* H3 pin sense - negative true */
    /* H2 pin sense - negative true */
    /* H1 pin sense - negative true */
    gmsv04 -> port_2.pgcr |= 0x20; /* enable H34 */
    gmsv04 -> port_2.psrr = 0x08; /* PC4 = PC4 */
    /* PC5 = PIRQ */
    /* PC6 = PC6 */
    gmsv04 -> port_2.pacr = 0x00;

    /* clear data avail flip flop */
    gmsv04 -> port_2.pbcr = 0x28; /* B submode X0 - double buffered input */
    /* H4 asserted */

    gmsv04 -> port_2.pcddr |= 0x93; /* make buffer control lines outputs */
    gmsv04 -> port_2.pcdr &= ~0x93; /* and make A input + B input */

    gmsv04 -> port_3.pgcr &= ~0x30; /* disable H12 and H34 */
    gmsv04 -> port_3.pgcr = 0x08; /* mode 0 (unidirectional 8-bit mode) */
    /* H34 disabled */
    /* H12 disabled */
    /* H4 pin sense - positive true */
    /* H3 pin sense - negative true */
    /* H2 pin sense - negative true */
    /* H1 pin sense - negative true */
    gmsv04 -> port_3.pgcr |= 0x20; /* enable H34 */
    gmsv04 -> port_3.psrr = 0x00;
    gmsv04 -> port_3.pacr = 0x00;

    /* disable remote electronics */
    gmsv04 -> port_3.pbcr = 0x20; /* B submode 00 - double buffered input */
}

```

```

/* H4 Output - negated */
gmsv04 -> port_3.pcdr |= 0x93; /* make buffer control lines outputs */
gmsv04 -> port_3.pcdr &= ~0x93; /* and make A input + B input */

gmsv04 -> debug.pcdr |= 0x10; /* make PC4 an output */
gmsv04 -> debug.pcdr &= ~0x10; /* turn off fail led */

gmsv04 -> intsel.vr_0 = vector;
gmsv04 -> intsel.cr_0 = 0x16; /* enable interrupts at level 6 */

return;
}

```

3.36 int1x.c

```

*-----*
#include <stdio.h>

#define putchar(c) fputc(c, stdout)

int1x(intarg)
int intarg;
{
    int jhi1, jhi2, jloi;

    if (intarg >= 0) {
        jloi = (intarg & 0x0f) + 48;
    } else {
        intarg = - intarg;
        jloi = (intarg & 0x0f) + 32;
    }

    jhi1 = (intarg >> 10) + 64;
    jhi2 = ((intarg >> 4) & 0x3f) + 64;
    if (jhi1 != 64) {
        putchar(jhi1);
        putchar(jhi2);
    } else if (jhi2 != 64) {
        putchar(jhi2);
    }
    putchar(jloi);
}

```

3.37 putc.c

```

*-----*
putc(pfio, c)
char *pfio;
int c;
{
    if (c > 0) {
        if (c == '\n')
            cpm(2, '\r');
        cpm(2, c);
    }
}

```

3.38 putl.c

```

*-----*
unsigned int putl(pfio, s, n)
char *pfio;
char *s;
unsigned int n;
{
    register unsigned int i;

    i = n;

    while (i-- != 0)
        putc(pfio, *s++);

    return n;
}

```

3.39 query.c

```

*-----*

```

```

#include <stdio.h>

query(prompt)
char *prompt;
{
    char buffer[81];
    int query;

    query = -1;
    do {
        fputs(prompt, stdout);
        fputs(" ? ", stdout);
        fgets(buffer, sizeof(buffer), stdin);
        if (buffer[0] == 'Y' || buffer[0] == 'y')
            query = 1;
        else if (buffer[0] == 'N' || buffer[0] == 'n')
            query = 0;
    } while (query < 0);
    return query;
}

```

3.40 queryi.c

```

*-----*
#include <stdio.h>
#define isdigit(x) ((x) >= '0' && (x) <= '9')

queryi(prefix, value, suffix, min, max)
char *prefix;
int *value;
char *suffix;
int min;
int max;
{
    auto char buffer[81];
    register int i;
    int itemp;
    int nondigit;

    do {
        nondigit = -1;
        itemp = *value;
        printf("%s = %d", prefix, *value);
        if (suffix[0] != '\0')
            printf(" %s", suffix);
        printf(" : ", stdout);
        fgets(buffer, sizeof(buffer), stdin);
        for (i = 0 ; i < sizeof(buffer) ; i++) {
            if (buffer[i] == '\n')
                break;
            if (!isdigit(buffer[i])) {
                buffer[0] = '\n';
                nondigit = 1;
                break;
            }
        }
        if (buffer[0] != '\n')
            itemp = atoi(buffer);
    } while (itemp < min || itemp > max || nondigit > 0);
    *value = itemp;
}

```

3.41 queryl.c

```

*-----*
#include <stdio.h>

queryl(prefix, value, suffix, min, max)
char *prefix;
long *value;
char *suffix;
long min;
long max;
{
    long atol();
    char buffer[81];
    long ltemp;

```

```

do {
    ltemp = *value;
    printf("%s = %ld", prefix, *value);
    if (suffix[0] != '\0')
        printf(" %s", suffix);
    printf(" : ");
    fgets(buffer, sizeof(buffer), stdin);
    if (buffer[0] != '\n')
        ltemp = atol(buffer);
} while (ltemp < min || ltemp > max);
*value = ltemp;
}

```

3.42 querys.c

```

*-----*
#include <stdio.h>

querys(prefix, string, length, suffix)
char *prefix;
char *string;
int length;
char *suffix;
{
    int actlen;
    char buffer[81];
    register int i;
    int maxlen;
    int newlen;

    if (length < 0) {
        maxlen = -length - 1;
        actlen = strlen(string);
    } else {
        maxlen = length;
        for (i = length - 1; i >= 0; i--) {
            if ((string[i] != ' ') && (string[i] != '\0'))
                break;
        }
        actlen = i + 1;
    }
    do {
        printf("%s = ", prefix);
        for (i = 0; i < actlen; i++)
            putchar(string[i]);
        if (suffix[0] != '\0')
            printf(" %s", suffix);
        printf(" : ");
        for (i = 0; i < maxlen; i++)
            putchar(' ');
        for (i = 0; i < maxlen; i++)
            putchar('\b');
        fgets(buffer, sizeof(buffer), stdin);
        for (i = 0; i < sizeof(buffer); i++)
            if (buffer[i] == '\n')
                break;
        newlen = i;
    } while (newlen > maxlen);
    if (newlen > 0) {
        for (i = 0; i < newlen; i++)
            string[i] = buffer[i];
        if (length < 0)
            string[i] = '\0';
        else
            for (; i < length; i++)
                string[i] = ' ';
    }
}

```

3.43 queryt.c

```

*-----*
#include <stdio.h>

queryt(label, string, length)
char *label;
char string[] [80];

```

```

int length;
{
    char buffer[81];
    register int i;
    register int j;
    int last_len;
    int line_len;
    int max_len;
    int nlm1;
    int num_lines;

    num_lines = (length + 79) / 80;
    nlm1 = num_lines - 1;
    last_len = length - nlm1 * 80;

    for (i = 0 ; i < num_lines ; i++) {
        printf("%s", label);
        if (num_lines > 1)
            printf(" (line %d of %d)", i + 1, num_lines);
        printf(" is currently :\n");
        if (i == nlm1)
            max_len = last_len;
        else
            max_len = 80;
        for (j = max_len - 1 ; j >= 0 ; j--)
            if (string[i][j] != ' ')
                break;
        line_len = j + 1;
        for (j = 0 ; j < line_len ; j++)
            putchar(string[i][j]);
        putchar('\n');
        for (j = 0 ; j < max_len ; j++)
            putchar(' ');
        putchar('\r');
        fgets(buffer, max_len + 1, stdin);
        if (buffer[0] != '\n') {
            /* user entered some new text */
            /* find the length of the new text */
            for (j = 0 ; j < max_len + 1 ; j++)
                if (buffer[j] == '\n')
                    break;
            line_len = j;
            /* copy new text into string */
            for (j = 0 ; j < line_len ; j++)
                string[i][j] = buffer[j];
            /* blank fill string */
            for (j = line_len ; j < 80 ; j++)
                string[i][j] = ' ';
        }
    }
}

```

3.44 realti.c

```

*-----*
/*****
/*
/* real time display
/*
/* This routine performs the real time quick look
/*
/* While this routine is running the following keys are recognized:
/*
/* ' ' - resets the full scale value to the current maximum
/* '\r' - exits real time display mode after current real time display
/*
/*****

#include <stdio.h>
#include "turbine.h"
#include "scsi.h"
#include "rt.h"

real_time_display()
{
    extern struct HEADER header; /* must be before sizeof(header.*) */
    extern struct OPSTAT opstat;

```

```

static int blade[MAX_RT_BLADES];
auto int c;
auto char cdb[6];
auto char date[10];
auto long data[MAX_RT_BLADES][MAX_RT_REVS];
register long *d_ptr;
static char excit_freq[sizeof(header.excit_freq) + 1];
auto int excit;
register int i;
register int j;
auto int id;
auto long max[MAX_RT_BLADES];
auto char message_in;
auto long min[MAX_RT_BLADES];
extern int n_blades;
auto long n_revs;
auto char prompt[14];
auto long range;
auto long rangemax;
extern long *rd_end;
extern long *rd_ptr;
extern long *rd_start;
auto long recip;
auto long revs_left;
extern int rtd_n_blades;
extern int rtd_n_revs;
static char run_id[sizeof(header.run_id) + 1];
auto unsigned int speed;
static int start_rev;
static int station;
auto char status;
auto long sum;
auto char time[9];
auto char temp;

if (opstat.data_avail && opstat.data_written)
    opstat.data_avail = FALSE;

rtd_n_revs = MAX_RT_REVS;

printf("\033[2J\033[1;1H"); /* Clear Screen and Home */

queryi("Station for real time display", &station, "", 1, 4);
id = 1 << (station - 1);

queryi("Number of blades to be displayed", &rtd_n_blades, "", 1, MAX_RT_BLADES);

for (i = 0; i < rtd_n_blades; i++) {
    sprintf(prompt, "Plot %d blade", i+1);
    queryi(prompt, &blade[i], "", 1, n_blades);
}

if (opstat.taking_data) {
    for (i = sizeof(header.run_id) - 1; i >= 0; i--)
        if (header.run_id[i] != ' ')
            break;
    run_id[i+1] = '\0';
    while (i >= 0) {
        run_id[i] = header.run_id[i];
        i--;
    }

    for (i = sizeof(header.excit_freq) - 1; i >= 0; i--)
        if (header.excit_freq[i] != ' ')
            break;
    excit_freq[i+1] = '\0';
    while (i >= 0) {
        excit_freq[i] = header.excit_freq[i];
        i--;
    }
} else {
    querys("Real time plot I.D.", run_id, -sizeof(run_id), "");
    querys("Excitation frequency", excit_freq, -sizeof(excit_freq), "");
}

scrn_init();

```



```

title(run_id, station, excit_freq);

rt_init(blade);

exit = FALSE;
rangemax = 0;
do {
    get_date(date);
    get_time(time);
    if (opstat.taking_data) {
        start_rev = rd_ptr - rd_start;
        revs_left = rd_end - rd_ptr;
        if (revs_left <= (rtd_n_revs + 2))
            break;
    } else if (opstat.data_avail) {
        /* we finished taking data - exit */
        break;
    } else {
        disable();
        rd_ptr = rd_start;
        n_revs = rtd_n_revs + 2; /* since first rev of data is bad */
        rd_end = rd_start + n_revs;
        setup(station, n_revs);
        enable();
        start_rev = 2; /* since first rev of data is bad */
    }
    if (scsi_sel(id)) {
        cdb[0] = REAL_TIME_DISPLAY;
        cdb[1] = start_rev >> 16;
        cdb[2] = start_rev >> 8;
        cdb[3] = start_rev;
        cdb[4] = rtd_n_blades;
        cdb[5] = rtd_n_revs;
        scsi_out(COMMAND, cdb, 6);
        /* send the blade numbers */
        for (i = 0 ; i < rtd_n_blades ; i++) {
            temp = blade[i];
            scsi_out(DATA_OUT, &temp, 1);
            min[i] = 0x7fffffff;
            max[i] = 0;
        }
        sum = 0;
        /* get the data */
        for (i = 0 ; i < rtd_n_revs ; i++) {
            d_ptr = rd_start + start_rev + i;
            /* wait for 1 ppr */
            while (d_ptr >= rd_ptr)
                ;
            sum += *d_ptr;

            for (j = 0 ; j < rtd_n_blades ; j++) {
                d_ptr = &data[j][i];
                scsi_in(DATA_IN, d_ptr, 4);
                if (*d_ptr < min[j])
                    min[j] = *d_ptr;
                if (*d_ptr > max[j])
                    max[j] = *d_ptr;
                range = max[j] - min[j];
                if (range > rangemax)
                    rangemax = range;
            }
        }
        scsi_in(STATUS, &status, 1);
        scsi_in(MESSAGE_IN, &message_in, 1);
        if (status == 0) {
            speed = (60 * 10000000) / (sum / rtd_n_revs);
            update(speed, date, time);
            rt_plot(data, rangemax, min, max);
        } else {
            printf("Error - data station had an error\n");
        }
    } else {
        printf("Error - data station does not respond\n");
    }
}
/* check if char available */
while (cpm(11, 0) != 0) {
    /* get char */

```

```

        c = cpm(1, 0);
        printf("\033LZ"); /* CLEAR DIALOG SCROLL */
        if (c == ' ')
            rangemax = 0;
        else if (c == '\r' || c == '\n')
            exit = TRUE;;
    }
} while (! exit);

printf("\033\014"); /* FARE */

printf("\033XX11"); /* Select ANSI mode */
}

```

3.45 rtinit.c

```

-----*
/*
/* rt_init
/*
/* This routine sets up and labels the screen for a real time display
/*
/*
*****/

#include <stdio.h>
#include "turbin.h"
#include "rt.h"

rt_init(blade)
int blade[];
{
    register int      i;
    extern int        rtd_hd2;
    extern int        rtd_height;
    extern int        rtd_n_blades;
    extern int        rtd_n_revs;
    extern int        rtd_xstep;
    extern int        rtd_ystep;
    extern int        rtd_ystart;
    auto int          x;
    auto int          y;

    rtd_ystep = YSIZE / rtd_n_blades;
    rtd_ystart = YSIZE - (rtd_ystep >> 1) + YOFFSET;
    rtd_xstep = XSIZE / (rtd_n_revs - 1);
    rtd_height = rtd_ystep - 2;
    rtd_hd2 = rtd_height >> 1;

    x = XOFFSET - 13;
    y = rtd_ystart;

    for (i = 0 ; i < rtd_n_blades ; i++) {
        /* draw border */
        cmd1x("LF");
        :y1x(XOFFSET, y-rtd_hd2);
        cmd1x("LG");
        xy1x(XOFFSET, y+rtd_hd2);
        cmd1x("LG");
        xy1x(XOFFSET + XSIZE, y+rtd_hd2);
        cmd1x("LG");
        xy1x(XOFFSET + XSIZE, y-rtd_hd2);
        cmd1x("LG");
        xy1x(XOFFSET, y-rtd_hd2);

        /* draw 0 line */
        cmd1x("LF");
        xy1x(XOFFSET, y);
        cmd1x("LG");
        xy1x(XOFFSET + XSIZE, y);

        /* print blade number */
        cmd1x("LF");
        xy1x(x, y - 3);
        cmd1x("LT");
        int1x(2);
        printf("%2d", blade[i]);
    }
}

```

```
    y -= rtd_ystep;
```

```
}
```

3.46 rtplot.c

```

*-----*
/*****
/*
/* rt_plot
/*
/* This routine plots the real time data on the screen
/*
/*
*****/

#include <stdio.h>
#include "turbin.h"
#include "rt.h"

#define move(x, y) cmd1x("LF"); xy1x(x, y);
#define draw(x, y) cmd1x("LG"); xy1x(x, y);

rt_plot(data, rangemax, min, max)
long data[MAX_RT_BLADES][MAX_RT_REVS];
long rangemax;
long max[];
long min[];
{
    auto    char    buffer[12];
    register long *d_ptr;
    auto    long    full_scale;
    register int    i;
    register int    j;
    auto    int    length;
    auto    long    mid;
    extern    int    rtd_hd2;
    extern    int    rtd_height;
    extern    int    rtd_n_blades;
    extern    int    rtd_n_revs;
    extern    int    rtd_xstep;
    extern    int    rtd_ystart;
    extern    int    rtd_ystep;
    auto    int    x;
    auto    int    ymid;

    full_scale = (rangemax + 1) >> 1;

    ymid = rtd_ystart;

    for (i = 0 ; i < rtd_n_blades ; i++) {
        /* update full scale - since plot above clears the char tops */
        /* label +/- full scales */
        move(XOFFSET - 61, ymid + rtd_hd2 - 7);
        cmd1x("LT");
        int1x(10);
        printf("%10d", full_scale);

        /* clear plot area */
        cmd1x("RR");
        xy1x(XOFFSET + 1, ymid - rtd_hd2 + 1);
        xy1x(XOFFSET + XSIZE - 1, ymid + rtd_hd2 - 1);
        int1x(0);

        mid = (max[i] + min[i]) >> 1;
        d_ptr = &data[i][0];

        move(XOFFSET, (*d_ptr++ - mid) * rtd_hd2 / full_scale + ymid);
        x = XOFFSET;
        for (j = 1 ; j < rtd_n_revs ; j++) {
            x += rtd_xstep;
            draw(x, (*d_ptr++ - mid) * rtd_hd2 / full_scale + ymid);
        }

        /* draw base line */
        move(XOFFSET, ymid);
        draw(XOFFSET + XSIZE, ymid);
    }
}

```

```

        move(XOFFSET - 67, ymid - rtd_hd2);
        cmd1x("LT");
        int1x(11);
        printf("%11d", -full_scale);

        ymid -= rtd_ystep;
    }
}

```

3.47 scrnin.c

```

*-----*
scrn_init()
{
    /* Clear text screen */
    printf("\033[2J");

    /* select TEK mode */

    printf("\033XX10");
    cmd1x("RW");
    xy1x(0, 0);
    xy1x(479, 359);

    /* Set GRAPHTEXT size */

    cmd1x("MC");
    int1x(5);
    int1x(7);
    int1x(6);

    /* Set Graphics Area Writing Mode */

    cmd1x("MG");
    int1x(0);
}

```

3.48 setdat.c

```

*-----*
/* this subroutine sets the date portion of the real time clock */
set_date(date)
char *date;
{
    unsigned char buffer[13];
    int day;
    int month;
    int year;

    clock_r(buffer);
    day = (buffer[8] & 0x03) * 10 + buffer[7];
    month = (buffer[10] & 0x01) * 10 + buffer[9];
    year = buffer[12] * 10 + buffer[11];
    queryi("Year", &year, "", 0, 99);
    queryi("Month", &month, "", 1, 12);
    queryi("Day", &day, "", 1, 31);
    clock_r(buffer);
    buffer[12] = year / 10;
    buffer[11] = year % 10;
    buffer[10] = month / 10;
    buffer[9] = month % 10;
    buffer[8] = day / 10;
    buffer[7] = day % 10;
    clock_w(buffer);
}

```

3.49 settim.c

```

*-----*
/* this subroutine sets the time portion of the real time clock */
set_time(time)
char *time;
{
    unsigned char buffer[13];
    int hour;
    int minute;
}

```

```

clock_r(buffer);
hour = (buffer[5] & 0x03) * 10 + buffer[4];
minute = (buffer[3] & 0x07) * 10 + buffer[2];
queryi("Hour", &hour, "", 0, 23);
queryi("Minute", &minute, "", 0, 59);
clock_r(buffer);
buffer[5] = hour / 10 + 0x0c;
buffer[4] = hour % 10;
buffer[3] = minute / 10;
buffer[2] = minute % 10;
buffer[1] = 0;
buffer[0] = 0;
clock_w(buffer);

```

}

3.50 setup.c

```
#include "scsi.h"
```

```
setup(station, n_revs)
```

```
int station;
```

```
long n_revs;
```

```
{
```

```

    char cdb[6];
    int id;
    char message_in;
    extern int n_blades;
    char status;

```

```

    cdb[0] = SETUP;
    cdb[1] = n_blades;
    cdb[2] = n_revs >> 16;
    cdb[3] = n_revs >> 8;
    cdb[4] = n_revs;
    cdb[5] = 0;
    id = 1 << (station - 1);
    if (scsi_sel(id)) {
        scsi_out(COMMAND, cdb, 6);
        scsi_in(STATUS, &status, 1);
        scsi_in(MESSAGE_IN, &message_in, 1);
    }

```

}

3.51 strlx.c

```
#include <stdio.h>
```

```
#define putchar(c) fputc(c, stdout)
```

```
strlx(len, string)
```

```
int len;
```

```
char *string;
```

```
{
```

```

    int lx(len);
    while (len > 0) {
        putchar(*string);
        string++;
    }

```

}

3.52 takeda.c

```
#include <stdio.h>
```

```
#include "turbine.h"
```

```
take_data()
```

```
{
```

```

    auto    char    date[10];
    extern  struct  HEADER  header;
    extern  struct  OPSTAT  opstat;
    register int    i;
    auto    long    max_revs;
    extern  int     n_blades;

```

```

static long      n_revs;
extern int       n_stations;
auto char       prompt[39];
extern long      *rd_end;
extern long      *rd_ptr;
extern long      *rd_start;
auto char       time[9];

header.max_speed = 0;
header.min_speed = 65535;
header.ave_speed = 0;
max_revs = (SLAVE_MEMSIZE >> 2) / n_blades - 1;
printf("\033[2J\033[1;1H"); /* Clear screen and home */
sprintf(prompt, "Number of revolutions (1 to %ld)", max_revs);
queryl(prompt, &n_revs, "", 1L, max_revs);
header.n_revs = n_revs;
querys("Run I.D.", header.run_id, sizeof(header.run_id), "");
queryt("Run description", header.run_des, sizeof(header.run_des));
querys("Excitation frequency", header.excit_freq,
        sizeof(header.excit_freq), "Hz.");
get_date(date);
strncpy(header.date, date, 9);
get_time(time);
strncpy(header.time, time, 8);
rd_end = rd_start + n_revs;
rd_ptr = rd_start;
disable();
for (i = 1 ; i <= n_stations ; i++) {
    setup(i, n_revs);
}
enable();
opstat.taking_data = TRUE;
opstat.data_avail = FALSE;
opstat.data_written = FALSE;
}

```

3.53 tapein.c

```

*-----*
#include <stdio.h>
#include "vme402.h"

tape_init()
{
    extern char last_op;

    *TAPE_CONTROL_STATUS_0 = 0xee;
    last_op = 0;
}

```

3.54 tapewr.c

```

*-----*
#include <stdio.h>
#include "vme402.h"

#define FALSE 0
#define TRUE 1

```

```

tape_write(buffer, length)
char buffer[];
int length;
{
    register short *dma_addr;
    register short *dma_data;
    register int error;
    extern char last_op;
    register int status;
    static short wchain[8];

    error = FALSE;

    status = *TAPE_COMMAND_STATUS_1;

    if ((status & RDY) != 0) {
        printf("ERROR - Transport not ready\n");
        error = TRUE;
    } else if ((status & ONL) != 0) {
        printf("ERROR - Transport off line\n");
        error = TRUE;
    } else if ((status & RWD) == 0) {
        printf("ERROR - Busy rewinding\n");
        error = TRUE;
    } else if ((status & FPT) == 0) {
        printf("ERROR - Tape write-protected\n");
        error = TRUE;
    }

    if (! error) {
        if (last_op == WRITE) {
            /* wait for 'Not in Data Transfer Mode' */
            while ((*TAPE_CONTROL_STATUS_0 & DBY) == 0)
                ;
        } else {
            /* wait for 'Formatter not busy' */
            while ((*TAPE_CONTROL_STATUS_0 & FBY) == 0)
                ;
        }

        status = *TAPE_CONTROL_STATUS_0;

        if ((status & DLOST) == 0 && (last_op == WRITE)) {
            printf("ERROR - Data lost during R/W\n");
            error = TRUE;
        }

        if ((status & EOT) != 0 && (last_op == WRITE)) {
            printf("ERROR - End Of Tape detected\n");
            error = TRUE;
        }

        if (! error) {
            /* set up dma controller */
            dma_addr = DMA_ADDR;
            dma_data = DMA_DATA;
            *dma_addr = CR;
            *dma_data = 0x00; /* Reset DMA controller */
            wchain[0] = 0x0382; /* register load mask */
            wchain[1] = (((short)((long)TAPE_DATA >> 8)) & 0xff00) | 0x10;
            wchain[2] = (short)TAPE_DATA;
            wchain[3] = (((short)((long)buffer >> 8)) & 0xff00) | 0x40;
            wchain[4] = (short)buffer;
            wchain[5] = length;
            wchain[6] = 0x0000; /* Channel mode 31:16 */
            wchain[7] = 0x0051; /* Channel mode 15:00 */ /* Bus Release */
            wchain[7] = 0x0031; /* Channel mode 15:00 */ /* Bus Hold */
            *dma_addr = MMR;
            *dma_data = 0x07;
            *dma_addr = CHAR_HI;
            *dma_data = (short)((long)wchain >> 8) & 0xff00;
            *dma_addr = CHAR_LO;
            *dma_data = (short)wchain;
            *dma_addr = CR;
            *dma_data = 0xa0; /* enable DMA transfer */

            /* send write command to tape drive */
            *TAPE_COMMAND_STATUS_1 = WRITE;
            last_op = WRITE;
        }
    }
}

```

```

        /* wait for 'In Data Transfer Mode' */
        while ((*TAPE_CONTROL_STATUS_0 & DBY) != 0)
            ;
    }
}

```

3.55 title.c

```

*-----*
title(run_id, station, excit_freq)
char *run_id;
int station;
char *excit_freq;
{
    cmd1x("LF");
    xy1x(0, 360-8);
    cmd1x("LT");
    int1x(54);

    printf("%16s    Station-X1d    %5s Hz          RPM", run_id, station,
           excit_freq);
}

```

3.56 turbin.c

```

*-----*
#include <stdio.h>
#include "turbin.h"

main()
{
    extern long      _memory;
    register int     c;
    extern int       cpm();
    extern struct HEADER header;
    auto long        i;
    auto long        old_rev;
    extern struct OPSTAT opstat;
    auto long        rd_base;
    extern long       *rd_end;
    extern long       *rd_ptr;
    extern long       *rd_start;
    auto long        rd_total;
    auto long        rev;
    extern void       r_isr();
    auto unsigned int speed;
    auto int          updated;

    *((long *) (VECTOR * 4)) = (long) (&r_isr);

    init_port(VECTOR);

    tape_init();

    printf("\033XX11"); /* Select ANSI mode */

    change();

    opstat.data_avail = FALSE;
    opstat.data_written = FALSE;
    opstat.taking_data = FALSE;

    for ( ; ; ) {
        updated = FALSE;
        printf("\033XX11"); /* Select ANSI mode */
        printf("\033[2J"); /* Clear Screen */
        printf("\033[1;7H\033#3NSMS Data Acquisition System");
        printf("\033[2;7H\033#4NSMS Data Acquisition System");
        printf("\033[10;20HC = Change Date / Time / Specimen Data");
        printf("\033[12;20HD = Discard Data");
        printf("\033[14;20HH = Histogram");
        printf("\033[16;20HR = Real Time Display");
        printf("\033[18;20HS = Start Taking Data");
        printf("\033[20;20HW = Write Data To Tape");
        printf("\033[22;20H");
        if (opstat.taking_data || opstat.data_avail) {

```



```

printf("\033[4;20H Revolution      of %d", header.n_revs);
if (opstat.taking_data)
    printf("\033[6;20H      Current      Minimum      Maximum");
else
    printf("\033[6;20H      Average      Minimum      Maximum");
printf("\033[7;20H      -----      -----      -----");
printf("\033[8;20H Speed :                                RPM");
}
while (cpm(11, 0) == 0) {
    if (opstat.taking_data) {
        rev = rd_ptr - rd_start;
        if (rev != old_rev) {
            printf("\033[4;31HX6ld", rev);
            old_rev = rev;
            if (rev > 2) {
                speed = (60 * 1000000) / (rd_start + rev - 1);
                printf("\033[8;28HX7lu", speed);
                if (speed < header.min_speed) {
                    header.min_speed = speed;
                    printf("\033[8;38HX7lu", header.min_speed);
                }
                if (speed > header.max_speed) {
                    header.max_speed = speed;
                    printf("\033[8;48HX7lu", header.max_speed);
                }
            }
        }
    }
}
if (opstat.data_avail && (header.ave_speed == 0)) {
    rd_total = 0;
    rd_base = rd_start[1];
    for (i = 1; i < header.n_revs; i++)
        rd_total = rd_total + rd_start[i] - rd_base;
    header.ave_speed = (60 * 1000000) / (rd_total / (header.n_revs-1) + rd_base);
    updated = FALSE;
}
if (! updated && (opstat.data_avail || opstat.taking_data)) {
    if (header.ave_speed != 0) {
        printf("\033[6;20H      Average      Minimum      Maximum");
        printf("\033[8;28HX7lu", header.ave_speed);
        printf("\033[4;31HX6ld", header.n_revs);
    }
    printf("\033[8;38HX7lu", header.min_speed);
    printf("\033[8;48HX7lu", header.max_speed);
    updated = TRUE;
}
}
while (cpm(11, 0) != 0) {
    c = cpm(1, 0);
    if (c == 'C' || c == 'c')
        change();
    else if (c == 'D' || c == 'd') {
        if (opstat.data_avail) {
            printf("\033[2J\033[1;1H");
            if (query("Are you sure you want to discard data"))
                opstat.data_avail = FALSE;
        }
    }
    else if (c == 'H' || c == 'h') {
        if (! opstat.data_avail || opstat.data_written)
            histogram();
    }
    else if (c == 'R' || c == 'r') {
        if (! opstat.data_avail || opstat.data_written)
            real_time_display();
    }
    else if (c == 'S' || c == 's') {
        if (! opstat.data_avail || opstat.data_written)
            take_data();
    }
    else if (c == 'W' || c == 'w')
        if (opstat.data_avail)
            write_data();
    }
}
}

```

3.57 twfm.c

```

*-----*
/* Write File Mark */

```

```
#include "vme402.h"

t_wfm()
{
    register char *status_0;
    extern char last_op;

    *TAPE_CONTROL_STATUS_0 = 0xee;

    /* check for 'Tape not write-protected' */
    if ((*TAPE_COMMAND_STATUS_1 & FPT) != 0) {
        status_0 = TAPE_CONTROL_STATUS_0;
        /* wait for 'Formatter not busy' */
        while ((*status_0 & FBY) == 0)
            ;
        last_op = T_WFM;

        /* send command to tape drive */
        *TAPE_COMMAND_STATUS_1 = T_WFM;
    }
}
```

3.58 update.c

```
*-----*
update(speed, date, time)
unsigned int speed;
char *date;
char *time;
{
    cmdlx("LF");
    xylx(45 * 6, 360-8);
    cmdlx("LT");
    intlx(54);

    printf("%5u RPM   %9s   %8s", speed, date, time);
}
```

3.59 writed.c

```
*-----*
#include <stdio.h>
#include "turbin.h"
#include "scsi.h"
#include "vme402.h"

#define BLOCKSIZE 512
#define BSD4 (BLOCKSIZE >> 2)

write_data()
{
    auto long buffer[2][BSD4];
    auto int block;
    auto char cdb[6];
    register long *d_start;
    register long *d_ptr;
    auto int error;
    auto int exit;
    auto int extra;
    extern struct HEADER header;
    extern struct OPSTAT opstat;
    register int i;
    register int j;
    register int k;
    auto int last_length;
    auto int length;
    auto char message_in;
    auto int n_blocks;
    auto long n_points;
    extern long *rd_end;
    extern long *rd_start;
    register long *s_ptr;
    auto char status;
    auto int status_1;
    auto char string[21];

    printf("\033[2J\033[1;1H");
}
```

```

do {
    exit = FALSE;
    if ((*TAPE_COMMAND_STATUS_1 & WRZ) == 0) {
        printf("Warning - tape not set at 1600 bpi\n");
        if (!query("Do you want to continue"))
            exit = TRUE;
    }
    error = FALSE;
    status_1 = *TAPE_COMMAND_STATUS_1;
    if ((status_1 & FPT) == 0) {
        printf("Error - tape is write protected\n");
        error = TRUE;
    }
    if ((status_1 & RWD) == 0) {
        printf("Error - tape is busy rewinding\n");
        error = TRUE;
    }
    if ((status_1 & OWL) != 0) {
        printf("Error - transport is off line\n");
        error = TRUE;
    }
    if ((status_1 & RDY) != 0) {
        printf("Error - transport is not ready\n");
        error = TRUE;
    }
    if (error && query("Do you want to exit"))
        exit = TRUE;
} while (error && !exit);

if (!exit) {
    printf("Writing header\n");
    s_ptr = &header;
    d_start = buffer;
    d_ptr = d_start;
    for (i = 0 ; i < BSD4 ; i++)
        *d_ptr++ = *s_ptr++;
    /* first part of header is 32 longs - reverse for VAX */
    reverse(d_start, 32);
    tape_write(d_start, BLOCKSIZE);

    printf("Writing revolution times\n");

    block = 0;
    for (s_ptr = rd_start ; s_ptr < rd_end ; ) {
        block++;
        printf("%d\n", block);
        /* switch to other buffer */
        d_start = &buffer[0][0] + BSD4 - (d_start - &buffer[0][0]);
        d_ptr = d_start;
        for (i = 0 ; i < BSD4 ; i++)
            if (s_ptr < rd_end)
                *d_ptr++ = *s_ptr++;
            else
                *d_ptr++ = 0;
        reverse(d_start, BSD4);
        tape_write(d_start, BLOCKSIZE);
    }

    n_points = header.n_revs * header.n_blades;
    n_blocks = n_points / BSD4;
    extra = n_points - n_blocks * BSD4;
    if (extra == 0) {
        last_length = BLOCKSIZE;
    } else {
        last_length = extra << 2;
        n_blocks = n_blocks + 1;
    }

    cdb[0] = RECEIVE;
    cdb[1] = 0;
    cdb[2] = 0;
    for (i = 0 ; i < header.n_stations ; i++) {
        printf("Writing blade times for station %d\n", i + 1);
        cdb[3] = BLOCKSIZE >> 8;
        cdb[4] = BLOCKSIZE;
        cdb[5] = 0x80; /* reset the receive data pointer */
        length = BLOCKSIZE;
    }
}

```

```

for (j = 1; j <= n_blocks; j++) {
    printf("%d\r", j);
    if (j == n_blocks) {
        length = last_length;
        cdb[3] = length >> 8;
        cdb[4] = length;
    }
    if (scsi_sel(1 << i)) {
        d_start = &buffer[0][0] + BSD4 - (d_start - &buffer[0][0]);
        scsi_out(COMMAND, cdb, 6);
        scsi_in(DATA_IN, d_start, length);
        scsi_in(STATUS, &status, 1);
        scsi_in(MESSAGE_IN, &message_in, 1);
        if (j == n_blocks)
            for (k = (last_length >> 2); k < BSD4; k++)
                d_start[k] = 0;
        reverse(d_start, BSD4);

        tape_write(d_start, BLOCKSIZE);
        cdb[5] = 0; /* don't reset the receive data pointer */
    }
}
t_wfm(); /* write file mark */
opstat.data_written = TRUE;
}

```

3.60 xylx.c

```

*-----*
#include <stdio.h>

#define putchar(c) fputc(c, stdout)

xylx(ix, iy)
int ix;
int iy;
{
    int keb, khix, khiy, klox, kloy;

    khiy = (iy >> 7) + 32;
    keb = (iy & 0x03) * 4 + (ix & 0x03) + 96;
    kloy = ((iy >> 2) & 0x1f) + 96;
    khix = (ix >> 7) + 32;
    klox = ((ix >> 2) & 0x1f) + 64;
    putchar(khiy);
    putchar(keb);
    putchar(kloy);
    putchar(khix);
    putchar(klox);
}

```

4.0 SLAVE PROGRAM SOFTWARE LISTINGS

4.1 a.s

```
*-----*
        .globl id
        .long 0x001000
        .long slave

id:     .byte 1
        .even
```

4.2 badcom.s

```
*-----*
        .globl bad_command

bad_command:

        moveq.l    #2,d0
        move.l d0,sense_data

        rts
```

4.3 badexc.s

```
*-----*
AmZ8536=0xfe0000

        .globl bad_exception

bad_exception:
        bset #3,AmZ8536+3 * turn on FAIL led
        rte
```

4.4 bladei.s

```
*-----*
*
*   blade_isr
*
*   This interrupt service routine handles the blade interrupts
*
*   At 12000 RPM (200 RPS) with 70 blades interrupts will occur
*   approximately every 71 microseconds.
*
gmsv04=    0xffff000

port2= 0x040
port3= 0x080
debug= 0x100

pbcrr= 0x0e+1
padr= 0x10+1
pbdr= 0x12+1
pcdr= 0x18+1

        .globl blade_isr

blade_isr:
        movem.l    d0/a0,-(sp)      * 24
        lea gmsv04,a0                * 12
        move.b port3+pbdr(a0),d0     * 12
        andi.w #0x000f,d0           * 8
        swap d0                      * 4
        movep.w    port2+padr(a0),d0 * 16
        move.l blade_d_ptr,a0        * 16
        move.l d0,(a0)+              * 8
        move.l a0,blade_d_ptr        * 16
        cmp.l blade_d_end,a0        * 22
*       blo.s 1f                      * 10
*       bcs.s 1f                      * 10
*       clear interrupt enable
        bclr #1,gmsv04+port2+pbcrr
1:       movem.l    (sp)+,d0/a0      * 28
```

```

*       rte                      * 24
*       ---
*       interrupt processing time 200
*       ---
*       total interrupt service time 246 cycles => 24.6 microseconds

```

4.5 dispat.s

```

*-----*
*
*   dispatch
*
*   call with:
*       d0.b = command byte
*
*
*       .globl dispatch
*
dispatch:
    lea    bad_command,a0
    and.w  #0x00ff,d0
    lsl.w  #2,d0
    cmpi.w #table_size,d0
*       bhs.s 1f
    bcc.s  1f
    lea    table,a0
    movea.l 0(a0,d0.w),a0
1:       jsr    (a0)
    rts

table: .long bad_command
       .long bad_command
       .long histogram
       .long request_sense
       .long bad_command
       .long bad_command
       .long real_time_display
       .long bad_command
       .long receive
       .long setup
       .long send
table_size= .-table

```

4.6 get1.s

```

*-----*
*
*   get1 - get 1 byte from the scsi port
*
*   returns:
*       d0.b = byte from scsi
*
SCSI=      0xfe0300
Current_Data=SCSI+0+0+1
Target_Command=  SCSI+3+3+1
Bus_and_Status=  SCSI+5+5+1

ASSERT_REQ-= 3
ACK-=      0

*       .globl get1
*
get1: bset  #ASSERT_REQ~,Target_Command
1:   btst  #ACK~,Bus_and_Status
    beq.s 1b

    move.b Current_Data,d0

    bclr  #ASSERT_REQ~,Target_Command
2:   btst  #ACK~,Bus_and_Status
    bne.s 2b

```

rts

4.7 get2.s

```
*-----*
*
*   get2 - get 2 bytes from the scsi port
*
*   returns:
*       d0.w = byte 1 : byte 2 from scsi
*
*   .globl get2
get2: bsr    get1
      lsl.w  #8,d0
      bsr    get1
      rts
```

4.8 get3.s

```
*-----*
*
*   get3 - get 3 bytes from the scsi port
*
*   returns:
*       d0.l = 0 : byte 1 : byte 2 : byte 3 from scsi
*
*   .globl get3
get3: clr.w  d0
      bsr    get1
      swap   d0
      bsr    get2
      rts
```

4.9 get4.s

```
*-----*
*
*   get4 - get 4 bytes from the scsi port
*
*   returns:
*       d0.l = byte 1 : byte 2 : byte 3 : byte 4 from scsi
*
*   .globl get4
get4: bsr    get2
      swap   d0
      bsr    get2
      rts
```

4.10 getn.s

```
*-----*
*
*   getn - get n bytes from the scsi port into a buffer
*
*   call with:
*       d0.l = number of bytes
*       a0 -> buffer
*
SCSI=      0xfe0300
Current_Data=SCSI+0*0+1
Target_Command=SCSI+3*3+1
```

```

Bus_and_Status=    SCSI+5+5+1
ASSERT_REQ~= 3
ACK~= 0

        .globl getn
getn: bset  #ASSERT_REQ~,Target_Command
1:  btst  #ACK~,Bus_and_Status
    beq.e 1b

    move.b Current_Data,(a0)+
    bclr  #ASSERT_REQ~,Target_Command
2:  btst  #ACK~,Bus_and_Status
    bne.s 2b

    subq.l #1,d0
    bne.s getn

    rts

```

4.11 histog.s

```

*-----*
* histogram
*
* 0 : 0 = command byte
* 1 : 3 = start rev #
* 4 : 5 = number of revs
*
* data bytes are as follows:
*
*      0 :      3 = rev_time for start rev - 1
*      4 :      7 = rev_time for start rev
*      8 :     11 = 2^47 / rev_time for start rev
*
* 8 * n + 4 : 8 * n + 7 = rev time for start rev + n
* 8 * n + 8 : 8 * n + 11 = 2^47 / rev_time for start rev + n
*
* for n = 1 to number of revs
*
SCSI= 0xfe0300
Target_Command=    SCSI+3+3+1
Data_Out= 0
Data_In= 1

        .globl histogram
*
* a0 -> minmax_buffer
* a1 -> data
* d1 = 2^47 / rev_time
* d6 = blade counter
* d7 = rev counter
*
histogram:
* get start rev # into d0.l
* bsr get3
* multiply by number of blades
* move.w d0,d5
* swap d0
* now start rev is split into d0.w : d5.w
* mulu n_blades,d0
* mulu n_blades,d5
* swap d0
* clr.w d0
* add.l d5,d0
* times 4 for long
* lsl.l #2,d0
* lea memory,a1
* add.l d0,a1
*
* get number of revs

```



```

    bsr    get2
    move.w d0,n_h_revs

    move.w n_blades,d0
    subq.w #2,d0
    move.w d0,nbm2

*   initialize minmax buffer
    move.l #0x7fffffff,d5
    lea    minmax_buffer,a0
    move.w n_blades,d0
    subq.w #1,d0
0:   move.l d5,(a0)+
    clr.l  (a0)+
    dbf    d0,0b

    move.b #Data_Out,Target_Command

*   get rev time for rev before start rev
    bsr    get4
    move.l d0,rev_time

    move.w n_h_revs,d7
    subq.w #1,d7
1:   lea    minmax_buffer,a0
*
*   check first blade for back dither
*
    move.l (a1)+,d5
    cmp.l  (a1),d5
    bls.s 2f

*
*   correct for back dither
*
    sub.l  rev_time,d5

2:   bsr    get4
    move.l d0,rev_time
    get 2^47 / rev_time
    bsr    get4
    move.l d0,d1

    move.l d5,d0
    bsr    minmax

    move.w nbm2,d6
    beq.s 4f

    subq.w #1,d6
3:   move.l (a1)+,d0
    bsr    minmax
    dbf    d6,3b

4:   *
*   check last blade for forward dither
*
    move.l (a1)+,d0
    cmp.l  -8(a1),d0
*
    bhs.s 5f
    bcc.s 5f

*
*   correct for forward dither
*
    add.l  rev_time,d0
5:   bsr    minmax

    dbf    d7,1b

    move.b #Data_In,Target_Command

    lea    minmax_buffer,a0
    move.w n_blades,d5
    subq.w #1,d5
7:   move.l (a0)+,d0
    sub.l  d0,(a0)
    moveq.l #4,d0

```

```

ber    putn
dbf    d5,7b

clr.l  sense_data
clr.b  d0

rts

.bss

nbm2:   .=.+2
n_h_revs: .=.+2
recip:   .=.+4
rev_time: .=.+4

```

4.12 minmax.s

```

*-----*
*      minmax
*
*      call with:
*      a0 -> minmax buffer
*      d0 = new data
*      d1 = 2^47 / rev_time
*
*      updates:
*      a0
*
*      destroys:
*      d0/d2-d5/a2
*
*      .globl minmax

minmax:    bra.s 2f      * temporarily skip scaling operation
           move.l d0,a2
           tst.l d0
           bpl.s 1f
           neg.l d0
1:         move.w d0,d3      * 4
           swap d0           * 4
           move.w d0,d2      * 4
           move.w d1,d5      * 4
           swap d1           * 4
           move.w d1,d4      * 4
           swap d1
           A = d2.w          AB
           B = d3.w          x CD
           C = d4.w          ----
           D = d5.w          BD
           AD
           BC
           AC
           move.w d5,d0      * 4
           mulu d3,d5        * 40
           mulu d4,d3        * 40
           mulu d2,d0        * 40
           mulu d4,d2        * 40
           move.l d0,d4
           AC = d2.l
           BC = d3.l
           AD = d4.l
           BD = d5.l
           move.l d4,d0
           swap d0
           clr.w d0
           add.l d0,d5
           move.l d4,d0
           clr.w d0
           swap d0
           addx.l d0,d2
           move.l d3,d0
           swap d0
           clr.w d0
           add.l d0,d5
           move.l d3,d0
           clr.w d0
           swap d0

```

```

*      addx.l d0,d2
*      d2.l : d5.l = product (0 to 2^47)
*      move.w d2,d0
*      swap d0
*      swap d5
*      move.w d5,d0
*      d0.l = product >> 16
*      move.l a2,d2
*      bpl.s 2f
*      neg.l d0
2:      cmp.l (a0)+,d0
*      bge.s 3f
*      move.l d0,-4(a0)
3:      cmp.l (a0)+,d0
*      ble.s 4f
*      move.l d0,-4(a0)
4:      rts

```

4.13 put1.s

```

*-----*
*
*      put1 - put 1 byte to the scsi port
*
*      call with:
*      d0.b = byte to be output to scsi
*
SCSI=      0xfe0300

Current_Data=      SCSI+0+0+1
Initiator_Command= SCSI+1+1+1
Target_Command=      SCSI+3+3+1
Bus_and_Status=      SCSI+5+5+1

ASSERT_DATA_BUS=      0
ASSERT_REQ~=      3
ACK~=      0

        .globl put1

put1:    bset  #ASSERT_DATA_BUS,Initiator_Command

        move.b d0,Current_Data

        bset  #ASSERT_REQ~,Target_Command

1:      btst  #ACK~,Bus_and_Status
        beq.s 1b

        bclr  #ASSERT_REQ~,Target_Command

2:      btst  #ACK~,Bus_and_Status
        bne.s 2b

        bclr  #ASSERT_DATA_BUS,Initiator_Command

        rts

```

4.14 putn.s

```

*-----*
*
*      putn - put n bytes from a buffer to the scsi port
*
*      call with:
*      d0.l = number of bytes
*      a0 -> buffer
*
SCSI=      0xfe0300

Current_Data=      SCSI+0+0+1
Initiator_Command= SCSI+1+1+1
Target_Command=      SCSI+3+3+1
Bus_and_Status=      SCSI+5+5+1

```

```

ASSERT_DATA_BUS= 0
ASSERT_REQ-= 3
ACK-= 0

.globl putn
putn: bset #ASSERT_DATA_BUS,Initiator_Command
1:  move.b (a0)+,Current_Data
    bset #ASSERT_REQ-,Target_Command
2:  btst #ACK-,Bus_and_Status
    beq.s 2b
    bclr #ASSERT_REQ-,Target_Command
3:  btst #ACK-,Bus_and_Status
    bne.s 3b
    subq.l #1,d0
    bne.s 1b
    bclr #ASSERT_DATA_BUS,Initiator_Command
    rts

```

4.15 realti.s

```

*-----*
*
*  real_time_display
*
*  0 : 0 = command byte
*  1 : 3 = start rev #
*  4 : 4 = number of blades
*  5 : 5 = number of revs
*
*  data_out bytes are the blade numbers
*
*  data_in bytes are as follows:
*
*  0 : 3 = first selected blade time for start rev
*  4 : 7 = second selected blade time for start rev
*  .
*  .
*  x : x+3 = first selected blade time for start rev + 1
*  x+4 : x+7 = second selected blade time for start rev + 1
*  .
*  .
*
MAXBLADES= 70
SCSI= 0xfe0300
Target_Command= SCSI+3+3+1
Data_Out= 0
Data_In= 1

.globl real_time_display
real_time_display:
    move.w n_blades,d0
    lsl.w #2,d0
    move.w d0,nbt4
*   get start_rev # into d0.l
    bsr    get3
*   multiply by the number of blades
    move.w d0,d5
    swap  d0
*   now start_rev is split into d0.w : d5.w
    mulu  n_blades,d0
    mulu  n_blades,d5
    swap  d0

```

```

        clr.w d0
        add.l d5,d0
*       times 4 for long
        lsl.l #2,d0
        lea     memory,a5
        add.l d0,a5      * a5 -> start rev data

        clr.w d0
        bsr     get1
        move.w d0,rtd_n_blades

        clr.w d0
        bsr     get1
        move.w d0,rtd_n_revs

        move.b #Data_Out,Target_Command

        move.w rtd_n_blades,d1
        subq.w #1,d1
        lea     offset_table,a2
1:       clr.w d0
        bsr     get1
        subq.w #1,d0
*       times 4 for long offset
        lsl.w #2,d0
        move.w d0,(a2)+
        dbf     d1,1b

        move.b #Data_In,Target_Command

        move.w rtd_n_revs,d2
        subq.w #1,d2

2:       lea     offset_table,a2

        move.w rtd_n_blades,d1
        subq.w #1,d1

3:       move.w (a2)+,d0      * get offset
        lea     0(a5,d0.w),a0

*       wait until data is available
4:       cmpa.l blade_d_ptr,a0
*       bhs.s 4b
        bcc.s 4b

        moveq.l #4,d0
        bsr     putn

        dbf     d1,3b

        adda.w nbt4,a5      * bump pointer to next rev's data

        dbf     d2,2b

        clr.l sense_data

        clr.b d0

        rts

        .bss

nbt4:    .+=2
offset_table: .+=MAXBLADES+MAXBLADES
rtd_n_blades: .+=2
rtd_n_revs:  .+=2

```

4.16 receiv.s

```

*-----*
*
*       receive
*
*       if byte 5 bit 7 = 1 then reset rcv_ptr to start of memory
*

```

```

SCSI= 0xfe0300
Target_Command= SCSI+3+3+1
Data_In= 1
        .globl receive
receive: bsr get1
        tst.b d0
        bne.s 3f

        bsr get3
        move.l d0,a0

        bsr get1
        lsl.b #1,d0
        bne.s 3f

        bcc.s 1f

        move.l #__memory,rcv_ptr
1:      move.l a0,d0
        beq.s 2f

        move.b #Data_In,Target_Command

        movea.l rcv_ptr,a0
        bsr putn
        move.l a0,rcv_ptr

2:      clr.l d0
        move.l d0,sense_data
        bra.s 4f

3:      moveq.l #2,d0
        move.l d0,sense_data

4:      rts

        .bss

rcv_ptr: . = .+4

```

4.17 request.s

```

*-----*
        .globl request_sense

SCSI= 0xfe0300
Target_Command= SCSI+3+3+1
Data_Out= 0

request_sense:
        bsr get1
        tst.b d0
        bne.s 2f

        bsr get1
        tst.b d0
        bne.s 2f

        bsr get1
        tst.b d0
        bne.s 2f

        bsr get1
        tst.b d0
        bne.s 1f
        move.b #4,d0
1:      clr.w d1
        move.b d0,d1

        bsr get1

```

```

        tst.b d0
        bne.s 2f

        move.b #Data_Out,Target_Command

        lea     sense_data,a0
        moveq.l #4,d0
        bsr     putn

        clr.l d0
        move.l d0,sense_data
        bra.s 3f

2:      moveq.l #2,d0
        move.l d0,sense_data

3:      rts

```

4.18 send.s

```

*-----*
*
*      send
*
SCSI= 0xfe0300

Target_Command= SCSI+3+3+1

Data_Out= 0

        .globl send

send:   bsr     get1
        tst.b d0
        bne.s 2f

        bsr     get3
        move.l d0,a0

        bsr     get1
        tst.b d0
        bne.s 2f

        move.l a0,d0
        beq.s 1f

        move.b #Data_Out,Target_Command

        lea     __memory,a0
        bsr     getn

1:      clr.l d0
        move.l d0,sense_data
        bra.s 3f

2:      moveq.l #2,d0
        move.l d0,sense_data

3:      rts

```

4.19 setup.s

```

*-----*
*      setup - handle scsi setup command packet
*
*      0 : 0 = command byte
*      1 : 1 = number of blades
*      2 : 4 = number of revs
*      5 : 5 = reserved
*
MEMEND= 0x1fffff

gmsv04= 0xffff000

port_2= gmsv04+0x040
port_3= gmsv04+0x080

```

```

*      PIT offsets

pbcr= 7+7+1
padr= 8+8+1
pbdr= 9+9+1
psr= 13+13+1

      .globl setup

setup: clr.w d0
      bsr get1
      move.w d0,n_blades
      bsr get3
      move.l d0,n_revs

      bsr get1

      lea _memory,a0
      move.l a0,blade_d_ptr

      move.l n_revs,d0
      move.w d0,d1
      swp d0
      mulu n_blades,d0
      swp d0
      clr.w d0
      mulu n_blades,d1
      add.l d1,d0
      lsl.l #2,d0
      add.l d0,a0
      move.l a0,blade_d_end

*      check to see if request will overflow memory
      cmp.l #MEMEND,a0
      bhi.s 8f

*      Port B Submode = Submode X0 (double buffered input)
*      H4 asserted (clear data available FF)
      move.b #0x28,port_2+pbcr

*      clear ports
1:      btst #2,port_3+psr
      beq 2f
      tst.b port_3+pbdr
      bra.s 1b
2:      btst #2,port_2+psr
      beq 3f
      tst.b port_2+padr
      tst.b port_2+pbdr
      bra.s 2b
3:
*      Port B Submode = Submode X0 (double buffered input)
*      H4 Control = Output pin - pulsed input handshake protocol
*      H4 Interrupt Enable = The H4 interrupt is disabled
*      H3 SVCRRQ Enable = The H3 interrupt and DMA request are enabled
*      H3 Status Control = The H3S status bit is set anytime input data is
*                          present in the double-buffered input path.
      move.b #0x3a,port_2+pbcr

      clr.l sense_data
      clr.b d0
      bra.s 9f

8:      move.l #0x01000000,sense_data
      moveq.l #0x02,d0

9:      rts

```

4.20 slave.s

```

*-----*
*
*      slave
*
*      this program implements the slave station
*

```



```

VECTOR=      64

*      GMSV06 port addresses

SCSI= 0xfe0300

Current_Data=   SCSI+0+0+1
Initiator_Command= SCSI+1+1+1
Mode=          SCSI+2+2+1
Target_Command= SCSI+3+3+1
Current_Bus_Status= SCSI+4+4+1

Command=      2
Status=      3
Message_In=   7

COMMAND_COMPLETE= 0

SEL~=      1

TARGET_MODE=  6

        .globl slave

slave:
*      clear memory - takes about 1 second
*      we can't do this in a subroutine because the return address would
*      be cleared also

        lea    0x000000,a0
        move.w #8-1,d1
        move.w #65536-1,d0
1:      clr.l  (a0)+
        dbf    d0,1b
        dbf    d1,1b

*      setup configuration port

        bsr    v06_init

*      setup all vectors to point to bad_exception routine

        move.w #256-1,d0
        lea    bad_exception,a0
        suba.l a1,a1
2:      move.l a0,(a1)+
        dbf    d0,2b

*      set up interrupt vector
        clr.l  d0
        move.b #VECTOR,d0
        lsl.w  #2,d0
        move.l d0,a0
        move.l #blade_isr,(a0)

        move.b #VECTOR,d0
        bsr    v04_init

        bset   #TARGET_MODE,Mode    * set target mode bit

        mtsr   #0x2000              * set interrupt mask to 0

3:      move.b id,d0

4:      btst   #SEL~,Current_Bus_Status
        beq.s  4b

        cmp.b  Current_Data,d0
        bne.s  4b

        move.b #0x08,Initiator_Command    * set busy

        move.b #Command,Target_Command

        bsr    get1

        bsr    dispatch

```

```

        move.b #Status,Target_Command

        bsr    put1

        move.b #Message_In,Target_Command

        move.b #COMMAND_COMPLETE,d0

        bsr    put1

        move.b #0x00,Target_Command      * clear MSG~, C_D- and I_O-
        move.b #0x00,Initiator_Command   * clear BSY~

        bra    3b

        .bss

        .globl blade_d_end
blade_d_end: .=.+4
        .globl blade_d_ptr
blade_d_ptr: .=.+4
        .globl minmax_buffer
minmax_buffer: .=.+560 * MAXBLADES * 4 * 2
        .globl n_blades
n_blades: .=.+2
        .globl n_revs
n_revs: .=.+4
        .globl sense_data
sense_data: .=.+4

```

4.21 v04ini.s

```

*-----*
*
*      v04_init
*
*      call with:
*          d0.b = interrupt vector
*
*
*      GMSV04 port base address
gmsv04=    0xffff000
*
*      GMSV04 port offsets
port_1=    0x000
port_2=    0x040
port_3=    0x080
config=    0x0c0
debug= 0x100
intsel=    0x140
*
*      PIT offsets
pgcr= 0+0+1
psrr= 1+1+1
paddr= 2+2+1
pbddr= 3+3+1
pcddr= 4+4+1
pivr= 5+5+1
pacr= 6+6+1
pbcr= 7+7+1
padr= 8+8+1
pbdr= 9+9+1
paar= 10+10+1
pbar= 11+11+1
pcdr= 12+12+1
psr= 13+13+1
*
*      68153 offsets
cr_0= 0+0+1
cr_1= 1+1+1
cr_2= 2+2+1
cr_3= 3+3+1
vr_0= 4+4+1
vr_1= 5+5+1
vr_2= 6+6+1

```

vr_3= 7+7+1

.globl v04_init

v04_init:

```
*      setup ports on GMSV04 card
lea    0xffff000,a0

*      disable H12 and H34
and.b  #0xcf,port_2+pgcr(a0)

*      mode 1 (unidirectional 16-bit mode)
*      H34 disabled
*      H12 disabled
*      H4 pin sense - negative true
*      H3 pin sense - negative true
*      H2 pin sense - negative true
*      H1 pin sense - negative true
move.b #0x40,port_2+pgcr(a0)

*      enable H34
bset   #5,port_2+pgcr(a0)

*      PC4 = PC4
*      PC5 = PIRQ-
*      PC6 = PC6
move.b #0x08,port_2+psrr(a0)

clr.b  port_2+pcr(a0)

*      B submode X0 - double buffered input
*      H4 asserted (clear data avail flip flop)
move.b #0x28,port_2+pbcr(a0)

*      make buffer control lines outputs
ori.b  #0x93,port_2+pcddr(a0)

*      and make A input + B input
andi.b #0x6c,port_2+pcdr(a0)

*      disable H12 and H34
andi.b #0xcf,port_3+pgcr(a0)

*      mode 0 (unidirectional 8-bit mode)
*      H34 disabled
*      H12 disabled
*      H4 pin sense - positive true
*      H3 pin sense - negative true
*      H2 pin sense - negative true
*      H1 pin sense - negative true
move.b #0x08,port_3+pgcr(a0)

*      enable H34
bset   #5,port_3+pgcr(a0)

clr.b  port_3+psrr(a0)
clr.b  port_3+pcr(a0)

*      B submode 00 - double buffered input
*      H4 Output - negated (disable remote electronics)
move.b #0x20,port_3+pbcr(a0)

*      make buffer control lines outputs
ori.b  #0x93,port_3+pcddr(a0)
*      and make A input + B input
andi.b #0x6c,port_3+pcdr(a0)

*      make PC4 an output
ori.b  #0x10,debug+pcddr(a0)
*      turn off fail led
andi.b #0xef,debug+pcdr(a0)

move.b d0,intsel+vr_0(a0)

*      enable interrupts at level 6
move.b #0x16,intsel+cr_0(a0)
```

rts

4.22 v06ini.s

*
* v06_init
*

* GMSV06 port addresses

Am28536=0xfe0000

.globl v06_init

v06_init:

* setup configuration port

```
lea Am28536,a0
move.b 7(a0),d0
nop
nop
move.b #0,7(a0)
nop
nop
move.b 7(a0),d0
move.w #tablesize-1,d0
lea table,a1
2: move.b (a1)+,7(a0)
dbf d0,2b
```

rts

```
table: .byte 0,1,0
       .byte 0x05,0x00 * Port C Data Path Polarity
       .byte 0x06,0x0e * Port C Data Direction
       .byte 0x07,0x00 * Port C Special I/O Control
       .byte 0x08,0x00 * Port A Command and Status
       .byte 0x09,0x00 * Port B Command and Status
       .byte 0x20,0x00 * Port A Mode Specification
       .byte 0x21,0x00 * Port A Handshake Specification
       .byte 0x22,0x00 * Port A Data Path Polarity
       .byte 0x23,0x08 * Port A Data Direction
       .byte 0x24,0x00 * Port A Special I/O Control
       .byte 0x28,0x00 * Port B Mode Specification
       .byte 0x29,0x00 * Port B Handshake Specification
       .byte 0x2a,0x00 * Port B Data Path Polarity
       .byte 0x2b,0xb0 * Port B Data Direction
       .byte 0x2c,0x00 * Port B Special I/O Control
       .byte 0x1e,0x95 * Counter/Timer 3 Mode Specification
       .byte 0x0d,0xee * Port A Data PA7 : NMEN* = 1
                       * PA6 : RAMCO = 1
                       * PA5 : MAPRO* = 1
                       * PA4 : WAIT0 = 0
                       * PA3 : PROM3 = 1
                       * PA2 : PROM2 = 1
                       * PA1 : PROM1 = 1
                       * PA0 : PROM0 = 0
       .byte 0x0e,0x47 * Port B Data PB7 : HALT* = ?
                       * PB6 : RESDIS* = 1
                       * PB5 : RESERVED = ?
                       * PB4 : SYSFAIL* = ?
                       * PB3 : FAIL = 0
                       * PB2 : RELES = 1
                       * PB1 : BUSL1 = 1
                       * PB0 : BUSLO = 1
       .byte 0x0f,0xee * Port C Data PC3 : CONTL = ?
                       * PC2 : DS = ?
                       * PC1 : CONTRL = ?
                       * PC0 : TIMOUT = 1
       .byte 0x01,0x94

tablesize= .-table
```

5.0 SIMULATOR EPROM LISTINGS

[illegible]

66

67

68

69

70

71

72

73

74

75

76

77

!

80